CSE 216 (Spring 2023) – Homework I

Instructor: Dr. Ritwik Banerjee

This homework requires you to use OCaml as the programming language, and is about **recursion & higher-order functions** as used in a functional programming paradigm. The document consists of 3 pages. Carefully read the entire document before you start coding.

Note 1: All functions, unless otherwise specified, should be polymorphic (i.e., they should work with any data type). For example, if you are writing a method that should work for lists, the type must be `'a list`, and not `int list`.

Note 2: You may not use any functions available in the OCaml library that already solves all or most of the question. For example, OCaml provides a `List.rev` function, but you may not use that in this section. You should also avoid the higher-order functions `List.map`, `List.fold_left`, and `List.filter`. You can, however, use helper functions that you have defined and written yourself. For example, you can define your own higher-order function called "map", and then use that to solve, say, problem 7.

1. Write a recursive function `pow`, which takes two integer parameters `x` and `n`, and returns $\texttt{x}^{\texttt{n}}$. Also write a function `float_pow`, which does the same thing, but for `x` being a float. You can assume that `n` is non-negative integer for both functions. (10)

2. Write a function `compress` to remove consecutive duplicates from a list. (10)

```
# compress ["a";"a";"b";"c";"c";"a";"a";"d";"e";"e";"e"];;
- : string list = ["a"; "b"; "c"; "a"; "d"; "e"]
```

3. A predicate is a function that returns a boolean value. In OCaml, the type of such a function can be expressed as `'a -> bool`. (10)

   Write a function `remove_if` of the type `'a list -> ('a -> bool) -> 'a list`, which takes a list and a predicate, and removes all the elements that satisfy the condition expressed in the predicate.

```
# remove_if [1;2;3;4;5] (fun x -> x mod 2 = 1);;
- : int list = [2; 4]
```

4. Some programming languages (like Python) allow us to quickly *slice* a list based on two integers `i` and `j`, to return the sublist from index `i` (inclusive) and `j` (not inclusive). We want such a slicing function in OCaml as well. (10)

   Write a function `slice` as follows: given a list and two indices, `i` and `j`, extract the slice of the list containing the elements from the $\texttt{i}^{\text{th}}$ (inclusive) to the $\texttt{j}^{\text{th}}$ (not inclusive) positions in the original list.

```
# slice ["a";"b";"c";"d";"e";"f";"g";"h"] 2 6;;
- : string list = ["c"; "d"; "e"; "f"]
```

   Invalid index arguments should be handled *gracefully*. For example,

```
# slice ["a";"b";"c";"d";"e";"f";"g";"h"] 3 2;;
- : string list = []
# slice ["a";"b";"c";"d";"e";"f";"g";"h"] 3 20;
- : string list = ["d";"e";"f";"g";"h"];
```

   You do *not*, however, need to worry about handling negative indices.

5. For this question, you will need to revisit the definition of an *equivalence* function from your prerequisite courses. (10)

   Write a function `equivs` of the type `('a -> 'a -> bool) -> 'a list -> 'a list list`, which partitions a list into equivalence classes according to a given equivalence function. Note that the equivalence function is provided as the first argument to `equivs`.

```
# equivs (=) [1;2;3;4];;
- : int list list = [[1];[2];[3];[4]]
# equivs (fun x y -> (=) (x mod 2) (y mod 2)) [1; 2; 3; 4; 5; 6; 7; 8];;
- : int list list = [[1; 3; 5; 7]; [2; 4; 6; 8]]
```

6. Goldbach's conjecture states that every positive even number greater than 2 is the sum of two prime numbers. E.g., $18 = 5 + 13$, or $42 = 19 + 23$. It is one of the most famous conjectures in number theory. It is unproven, but verified for all integers up to 4×10^{18}. Write a function `goldbachpair : int -> int * int` to find two prime numbers that sum up to a given even integer. The returned pair must have a non-decreasing order. (10)

```
# goldbachpair 10;;  (* can return (3, 7) but should not return (7, 3) *)
- : int * int = (3, 7)
```

   Note that the decomposition is not always unique. E.g., 10 can be written as 3+7 or as 5+5, so both `(3, 7)` and `(5, 5)` are correct answers.

7. Write a function called `identical_on`, which takes three inputs: two functions `f` and `g`, and a list `lst`. It returns `true` if and only if the functions `f` and `g` have identical behavior on every element of `lst`. (10)

```
# let f i = i * i;;
val f : int -> int = <fun>
# let g i = 3 * i;;
val g : int -> int = <fun>
# identical_on f g [3];;
- : bool = true
# identical_on f g [1;2;3];;
- : bool = false
```

8. Write a function called `pairwisefilter` with two parameters: (i) a function `cmp` that compares two elements of a specific type `T` and returns one of them, and (ii) a list `lst` of elements of that same type `T`. It returns a list that applies `cmp` while taking two items at a time from `lst`. If `lst` has odd size, the last element is returned "as is". (10)

```
# pairwisefilter min [14; 11; 20; 25; 10; 11];;
- : int list = [11; 20; 10]
# (* assuming that shorter : string * string -> string = <fun> already exists *)
# pairwisefilter shorter ["and"; "this"; "makes"; "shorter"; "strings"; "always"; "win"];;
- : string list = ["and"; "makes"; "always"; "win"]
```

9. Write the `polynomial` function, which takes a list of tuples and returns the polynomial function corresponding to that list. Each tuple in the input list consists of (i) the coefficient, and (ii) the exponent. You should assume that the tuples in the input list are given in decreasing order of the exponent. (10)

```
# (* below is the polynomial function f(x) = 3x^3 - 2x + 5 *)
(* note the decreasing order of exponents: x^3, then x^1, then x^0 *)
# let f = polynomial [3, 3; -2, 1; 5, 0];;
val f : int -> int = <fun>
# f 2;;
- : int = 25
```

10. The **power set** of a set S is the set of all subsets of S (including the empty set and the entire set). Write a function `powerset` of the type `'a list -> 'a list list`, which treats lists as unordered sets, and returns the powerset of its input list. You may assume that the input list has no duplicates. (10)

```
# powerset [3; 4; 10];; (* for sets, the item order does not matter *)
- : int list list = [[]; [3]; [4]; [10]; [3; 4]; [3; 10]; [4; 10]; [3; 4; 10]]
```

- Please keep in mind the homework-related points mentioned in the syllabus on our course web page!.
- **What to submit?** A single `.ml` file named `hw1.ml`, which should contain your code for all ten questions in this assignment.

  This assignment will be graded by a script, so be absolutely sure that you follow the above rule.
- Make sure that your `.ml` file actually compiles and runs. Do NOT simply check individual methods and submit the final code! You don't want to be in a situation where `hw1.ml` doesn't compile because of that one function you incorrectly copy-pasted from your terminal REPL!

Submission Deadline: Mar 3 (Friday), 11:59 pm