The overall objective of this assignment is for you to continue to gain experience with OCaml, adding in some complexity in terms of higher order functions (including anonymous functions) and tuples. All the problems require relatively little code ranging from 2 to 10 lines. If any function requires more than that, you can be sure that you need to rethink your solution. Download the handout from autolab and add your function definitions and comments to the misc3.ml file (it contains skeleton OCaml functions) Your task is to replace the text in those files with the appropriate OCaml code for each of those expressions. Do NOT change any of the function headers and be sure to write each function to the exact specs of the lab, including name. Your code will be auto tested, and any deviations will cause your code to fail testing.

To load your code, while in the utop or ocaml interpreter (run from the same directory as misc.ml), type: 
#use "misc3.ml"; **Be sure to test your file this way before uploading it to autolab. If utop throws errors, so will autolab.**

Note: All the solutions must be done using the purely functional fragment of OCaml, using constructs covered in class, and most require the use of recursion. Solutions using imperative features such as references, arrays, or while loops will receive no credit. Do not use any additional libraries, just use standard OCaml as installed in the Linux and Windows labs.

See the course website (resources page) for a few external sources on tuples and pattern matching.

Each problem should bind only one symbol at the top level. Do not litter the global namespace with helper functions. Nest them or define them inline as anonymous functions as appropriate.

1. (10 pts) Without using any built-in OCaml functions, write an OCaml function `assoc : int * string * (string * int) list -> int (or more generally, 'a * 'b * ('b * 'a) list -> 'a)` that takes a single parameter which is a tuple of three values `(d,k,l)` where `l` is a list of key-value pairs `[(k1,v1);(k2,v2);...]` and finds the first `k` that equals `k`. If such a `k` is found, then `v` is returned. Otherwise, the default value `d` is returned.

   Your function should be tail recursive. Once you have implemented the function, you should get the following behavior at the OCaml prompt:

   ```
   # assoc (-1,"jeff",[("sorin",85);("jeff",23);("jeff",44)]);
   - : int = 23
   # assoc (-1,"bob",[("sorin",85);("jeff",23);("moose",44);("meg",99)]);
   - : int = -1
   ```

2. (15 pts) Add code to the skeleton for removeDuplicates in misc3.ml to obtain a function of type `int list -> int list` that takes a list `l` and returns the list of elements of `l` with the duplicates, i.e. second, third, etc. occurrences, removed, and where the remaining elements appear in the
same order as in l. You should only replace the two occurrences of ‘failwith “to be written”’ in the existing skeleton. Do not change any of the other skeleton code.

For this problem only, you may use the library functions List.rev and List.mem. Once you have implemented the function, you should get the following behavior at the OCaml prompt:

```ocaml
# removeDuplicates [1;6;2;4;12;2;13;6;9];;
- : int list = [1;6;2;4;12;13;9]
```

3. (15 pts) Without using any built-in OCaml functions (and definitely not the while or for construct), write an OCaml function: wwhile : (int -> int * bool) * int -> int (or more generally, (a -> a * bool) * a -> a) that takes as input a tuple (f,b) and calls the function f on input b to get a pair (b',c'). wwhile should continue calling f on b' to update the pair as long as c' is true. Once f returns a c' that is false, wwhile should return b'.

Your function should be tail recursive. Once you have implemented the function, you should get the following behavior at the OCaml prompt:

```ocaml
# let f x = let xx = x*x*x in (xx,xx<100);
val f : int -> int * bool = fn
# wwhile (f,2);
- : int = 512
```

In other words, the function parameter (f in the above example) needs to be a function that takes an int and returns a tuple with an int and a bool. wwhile is a function that takes a tuple – a function and an int. wwhile applies the provided function to the int and gets back an int * bool tuple. As long as the bool in the result is true, wwhile continues to recurse by using the new int result.

4. (15 pts) Without using any built-in OCaml functions, modify the skeleton for fixpoint to obtain a function of type (int -> int) * int -> int (or more generally, (a -> a) * a -> a) which repeatedly updates b with f(b) until b=f(b) and then returns b. You should only replace the ‘failwith “to be written”’ in the existing skeleton. Do not change any of the other skeleton code.

Once you have implemented the function, you should get the following behavior at the OCaml prompt:

```ocaml
# let g x = truncate (1e6 *. cos (1e-6 *. float x));;
val f : int -> int = fn
# fixpoint (g,0);
- : int = 739085
```

In other words, fixpoint is using wwhile to find the value, x, of a function f where f(x) = x. To do this, you will need to write an anonymous function to pass in as part of the input to wwhile. In the first example above, the function being tested is g and the starting point for calling it is 0. g(739085) = 739085. This one is abstract and getting the pieces to fit together can be confusing, but it is essentially a one-liner.
Submission: Login to autolab and select the Lab 3 assignment. Upload and submit your revised version of misc3.ml.
Attribution: This assignment was (slightly) adapted from Sorin Lerner’s CSE 130 course at UCSD.