MULTISET (25/25 points)

A multiset is like a set, with the difference that a value can appear more than once.

1. Implement a module `MultiSet` that implements the signature `MultiSet_S`.
2. Define a function `letters: string -> char MultiSet.t` (where `MultiSet` is the module defined in the previous question). This function produces a multiset in which all characters are associated to the times they appear in the input string.
3. Define a function `anagram: string -> string -> bool` that uses the previous function to tell if two words have the same multiset of characters.

THE GIVEN PRELUDE

```
module type MultiSet_S = sig

  (* A multi-set of type ['a t] is a collection of values of
     type ['a] that may occur several times. *)
  type 'a t

  (* [occurrences s x] return the number of time [x] occurs
     in [s]. *)
  val occurrences : 'a t -> 'a -> int

  (* The empty set has no element. There is only one unique
     representation of the empty set. *)
  val empty : 'a t

  (* [insert s x] returns a new multi-set that contains all
     elements of [s] and a new occurrence of [x]. Typically,
     [occurrences s x = occurrences (insert s x) x + 1]. *)
  val insert : 'a t -> 'a -> 'a t

  (* [remove s x] returns a new multi-set that contains all elements
     of [s] minus an occurrence of [x] (if [x] actually occurs in
     [s]). Typically, [occurrences s x = occurrences (remove s x) x -
     1] if [occurrences s x > 0]. *)
  val remove : 'a t -> 'a -> 'a t

end
```

YOUR OCAML ENVIRONMENT

```
1  module MultiSet = struct
2    (* A multi-set of type ['a t] is a collection of values of
3       type ['a] that may occur several times. *)
4    type 'a t = 'a list
5
6    (* [occurrences s x] return the number of time [x] occurs
7       in [s]. *)
8    let rec occurrences l elt = match l with
9      | [] -> 0
10     | hd::tl ->  if hd = elt then  1 + occurrences tl elt else occurrences tl elt
11
12
13   (* The empty set has no element. There is only one unique
14      representation of the empty set. *)
15   let empty = []
16
17   (* [insert s x] returns a new multi-set that contains all
18      elements of [s] and a new occurrence of [x]. Typically,
19      [occurrences s x = occurrences (insert s x) x + 1]. *)
20   let insert l elt = elt::l
21
22   (* [remove s x] returns a new multi-set that contains all elements
23      of [s] minus an occurrence of [x] (if [x] actually occurs in
24      [s]). Typically, [occurrences s x = occurrences (remove s x) x -
25      1] if [occurrences s x > 0]. *)
26   let rec remove l elt =
27     let rec moins liste e res count = match liste with
28       | [] -> res
29       | hd::tl -> if e = hd && count = 0 then moins tl e res (count + 1) else
30             moins tl e (hd::res) count
31     in moins l elt [] 0
32
33 end ;;
```

Evaluate >

Switch >>

Typecheck

Reset Templ

Full-screen

Check & Sa

Found `MultiSet` with compatible type.
Your module is compatible with the `MultiSet` signature. 5 pts

v Exercise 2: `letters` Completed, 10 pts

Found `letters` with compatible type.
Computing `letters "4456"`
Correct value `[('4', 2); ('5', 1); ('6', 1)]` 1 pt
Computing `letters "- --#OCamlOCaml#ba"`
Correct value 1 pt
```
  [(' ', 1); ('#', 2); ('-', 3); ('C', 2); ('O', 2); ('a', 3); ('b', 1);
   ('l', 2); ('m', 2)]
```
Computing `letters "-"`
Correct value `[('-', 1)]` 1 pt
Computing `letters "OCP#OCP#-OCPOCPOCaml"`
Correct value 1 pt
```
  [('#', 2); ('-', 1); ('C', 5); ('O', 5); ('P', 4); ('a', 1); ('l', 1);
   ('m', 1)]
```
Computing `letters ", "`
Correct value `[(' ', 1); (',', 1)]` 1 pt
Computing `letters "ba, 4456-OCP, "`
Correct value 1 pt
```
  [(' ', 2); (',', 2); ('-', 1); ('4', 2); ('5', 1); ('6', 1); ('C', 1);
   ('O', 1); ('P', 1); ('a', 1); ('b', 1)]
```
Computing `letters "bebe//baOCPOCaml"`
Correct value 1 pt
```
  [('/', 2); ('C', 2); ('O', 2); ('P', 1); ('a', 2); ('b', 3); ('e', 2);
   ('l', 1); ('m', 1)]
```
Computing `letters "be#//-#4456, "`
Correct value 1 pt
```
  [(' ', 1); ('#', 2); (',', 1); ('-', 1); ('/', 2); ('4', 2); ('5', 1);
   ('6', 1); ('b', 1); ('e', 1)]
```
Computing `letters "4456, "`
Correct value `[(' ', 1); (',', 1); ('4', 2); ('5', 1); ('6', 1)]` 1 pt
Computing `letters ", "`
Correct value `[(' ', 1); (',', 1)]` 1 pt

v Exercise 3: `anagram` Completed, 10 pts

Found `anagram` with compatible type.
Computing `anagram ", OCaml4456baOCaml #OCPba" "4456OCPbeOCPOCamlba"`
Correct value `false` 1 pt
Computing `anagram "" ""`
Correct value `true` 1 pt
Computing `anagram "#, be#OCamlba, " "bCeal#a,#mOb ,"`
Correct value `true` 1 pt
Computing `anagram "OCaml//" "be, 4456"`
Correct value `false` 1 pt
Computing `anagram "be, ba //ba//beOCP" "OCaml#, ba//#4456#"`
Correct value `false` 1 pt
Computing `anagram "OCaml" "mClOa"`
Correct value `true` 1 pt
Computing `anagram "" ""`
Correct value `true` 1 pt
Computing `anagram " --4456OCP//-be" " //OCP "`
Correct value `false` 1 pt
Computing `anagram ", " ", "`
Correct value `true` 1 pt
Computing `anagram ", , , //" "be##bebaOCP"`
Correct value `false` 1 pt