SIMPLE USES OF REFERENCES (50/50 points)

1. Define `swap : 'a ref -> 'a ref -> unit` that swaps the contents of two references.
2. Define `update : 'a ref -> ('a -> 'a) -> 'a` that calls a function on the contents of a reference, updates it with the result, and returns the old value.
   For instance `let r = ref 6 in update r (function x -> x + 1)` should return `6` and set the reference to `7`.
3. Define `move: 'a list ref -> 'a list ref -> unit`, that removes the top argument from the first list and puts it on top of the second list. If the first list is empty, it should `raise Empty`.
4. A common pattern is to use a reference to perform a computation in an imperative way, but to keep it in a local definition, completely invisible from outside the function implementation.
   Define `reverse: 'a list -> 'a list`, that has a type and an observable behaviour that look like the ones of a purely functional function, buf that use a reference internally to perform the computation. It takes a list, and returns a copy of the list whose elements are in reverse order.
   The only functions you can call, except from locally defined functions, are `(!)`, `(:=)`, `ref`, and `move` that you just defined. And you are not allowed to use pattern matching.

THE GIVEN PRELUDE

```
exception Empty ;;
```

YOUR OCAML ENVIRONMENT

```
let swap ra rb =
  let rc = ref !rb in
  rb := !ra;
  ra := !rc
;;

let update r f =
  let res = ref !r in
  r := f !r;
  !res
;;

let move l1 l2 =
  match !l1 with
  | [] -> raise Empty
  | hd::tl ->
      l2 := hd :: !l2;
      l1 := tl
;;

let reverse l =
  let rev = ref [] in
  let liste = ref l in
  let doing =
    try
      while true do
        move liste rev
      done
    with _ -> ()
  in doing;
  !rev
;;
```

Evaluate >

Switch >>

Typecheck

Reset Templ

Full-screen

Check & Sa

Exercise complete (click for details) 50 pts

v Exercise 1: swap — Completed, 10 pts

Found swap with compatible type.
Computing `swap (ref 'f') (ref 'o')`
Correct value `((ref 'o'), (ref 'f'))` — 1 pt
Computing `swap (ref 'w') (ref 'f')`
Correct value `((ref 'f'), (ref 'w'))` — 1 pt
Computing `swap (ref 's') (ref 's')`
Correct value `((ref 's'), (ref 's'))` — 1 pt
Computing `swap (ref 'l') (ref 'm')`
Correct value `((ref 'm'), (ref 'l'))` — 1 pt
Computing `swap (ref 'h') (ref 'i')`

Computing `swap (ref 'h') (ref 'p')`
Correct value `((ref 'p'), (ref 'h'))` 1 pt
Computing `swap (ref 'z') (ref 'y')`
Correct value `((ref 'y'), (ref 'z'))` 1 pt
Computing `swap (ref 'k') (ref 'k')`
Correct value `((ref 'k'), (ref 'k'))` 1 pt
Computing `swap (ref 'n') (ref 'j')`
Correct value `((ref 'j'), (ref 'n'))` 1 pt

v Exercise 2: `update` Completed, 20 pts

Found `update` with compatible type.
Computing `update (ref 1) pred`
Correct value `1` 1 pt
Correct value `(ref 0)` 1 pt
Computing `update (ref 2) succ`
Correct value `2` 1 pt
Correct value `(ref 3)` 1 pt
Computing `update (ref -4) succ`
Correct value `-4` 1 pt
Correct value `(ref -3)` 1 pt
Computing `update (ref 1) pred`
Correct value `1` 1 pt
Correct value `(ref 0)` 1 pt
Computing `update (ref 4) (fun x -> x * 2)`
Correct value `4` 1 pt
Correct value `(ref 8)` 1 pt
Computing `update (ref 2) (fun x -> x * 2)`
Correct value `2` 1 pt
Correct value `(ref 4)` 1 pt
Computing `update (ref 1) succ`
Correct value `1` 1 pt
Correct value `(ref 2)` 1 pt
Computing `update (ref 3) succ`
Correct value `3` 1 pt
Correct value `(ref 4)` 1 pt
Computing `update (ref -3) pred`
Correct value `-3` 1 pt
Correct value `(ref -4)` 1 pt
Computing `update (ref 1) (fun x -> x * 2)`
Correct value `1` 1 pt
Correct value `(ref 2)` 1 pt

v Exercise 3: `move` Completed, 10 pts

Found `move` with compatible type.
Computing `move (ref [3; -1; -5; 2; 2; -2; -1; 0]) (ref [1; 3; -2])`
Correct value `((ref [-1; -5; 2; 2; -2; -1; 0]), (ref [3; 1; 3; -2]))` 1 pt
Computing `move (ref [3]) (ref [-2; -1; -3; -1])`
Correct value `((ref []), (ref [3; -2; -1; -3; -1]))` 1 pt
Computing `move (ref [0]) (ref [2; -5; -1; -5; 3; 3; 1; -4; -4; 3])`
Correct value `((ref []), (ref [0; 2; -5; -1; -5; 3; 3; 1; -4; -4; 3]))` 1 pt
Computing `move (ref [-2]) (ref [-4])`
Correct value `((ref []), (ref [-2; -4]))` 1 pt
Computing `move (ref []) (ref [3])`
Correct exception `Empty` 1 pt
Computing `move (ref [3; -2; 2; -4]) (ref [0; 2])`
Correct value `((ref [-2; 2; -4]), (ref [3; 0; 2]))` 1 pt
Computing `move (ref [3; -5; 4; -3; 0; -5; -3; -3]) (ref [])`
Correct value `((ref [-5; 4; -3; 0; -5; -3; -3]), (ref [3]))` 1 pt
Computing `move (ref [-3; -3; -1]) (ref [2; -4])`
Correct value `((ref [-3; -1]), (ref [-3; 2; -4]))` 1 pt
Computing `move (ref []) (ref [-4; -4; 0; 4; -4; -2; 0; 4; -5; 0])`
Correct exception `Empty` 1 pt
Computing `move (ref [0; 2; 3; 2]) (ref [])`
Correct value `((ref [2; 3; 2]), (ref [0]))` 1 pt

Found reverse with compatible type:
Computing reverse ['i'; 'n'; 'c']
Correct value ['c'; 'n'; 'i'] 1 pt
Computing reverse ['e'; 'u'; 'q'; 'i'; 'k'; 'r']
Correct value ['r'; 'k'; 'i'; 'q'; 'u'; 'e'] 1 pt
Computing reverse []
Correct value [] 1 pt
Computing reverse ['a'; 'w'; 'p'; 'z'; 'y'; 'a'; 'c'; 'v'; 'f']
Correct value ['f'; 'v'; 'c'; 'a'; 'y'; 'z'; 'p'; 'w'; 'a'] 1 pt
Computing reverse ['c'; 't']
Correct value ['t'; 'c'] 1 pt
Computing reverse ['b'; 'n'; 'q']
Correct value ['q'; 'n'; 'b'] 1 pt
Computing reverse ['q'; 'g'; 'e'; 'r'; 'b'; 'g'; 'r'; 'w'; 'w']
Correct value ['w'; 'w'; 'r'; 'g'; 'b'; 'r'; 'e'; 'g'; 'q'] 1 pt
Computing reverse ['d'; 'q'; 'h'; 'v'; 'c']
Correct value ['c'; 'v'; 'h'; 'q'; 'd'] 1 pt
Computing reverse []
Correct value [] 1 pt
Computing reverse ['v'; 'y'; 'n'; 'w'; 'g'; 'l'; 'k'; 'j'; 'u'; 'o']
Correct value ['o'; 'u'; 'j'; 'k'; 'l'; 'g'; 'w'; 'n'; 'y'; 'v'] 1 pt