COMP 3958: Lab 1

Put your implementation in a file named `lab1.ml`. Your file must compile without warnings or errors. If it does not, you may receive no credit for this lab. Maximum score: 14.

Implement each of the following functions using recursion without calling any external function except those in `Stdlib` and the `List.rev` function. For `zip`, `unzip` and `dedup`, provide both tail-recursive and non-tail-recursive implementations. The name of the tail-recursive version should end in `_tr`, e.g., the two "zip" functions should be named `zip` and `zip_tr`. You may implement additional helper functions if necessary. Provide at least 3 tests for each implementation.

1. `val zip : 'a list -> 'b list -> ('a * 'b) list`

   `zip lst1 lst2` is the list formed by pairing corresponding elements of `lst1` and `lst2` into a tuple. If `lst1` and `lst2` are of different lengths, pairing stops when the shorter list ends. For example,

   `zip [1; 2; 3] ['a'; 'b'; 'c'; 'd']` returns `[(1,'a'); (2,'b'); (3,'c')]`.

2. `val unzip : ('a * 'b) list -> 'a list * 'b list`

   `unzip lst`, where `lst` is a list of pairs, is a pair of lists where the first list consists of the first component of each pair in `lst` and the second list consists of the second component of each pair in `lst`. For example,

   `unzip [(1,'a'); (2,'b'); (3,'c')]` returns `([1; 2; 3], ['a'; 'b'; 'c'])`

3. `val dedup: 'a list -> 'a list`

   `dedup lst` is the list formed from `lst` by collapsing consecutive duplicated elements into a single element. For example,

   `dedup [1; 1; 2; 3; 3; 3; 2; 1; 1]` returns `[1; 2; 3; 2; 1]`

4. `val split_last: 'a list -> ('a list * 'a) option`

   `split_last lst` returns a pair whose first component is the first (n - 1) elements of `lst` (where n is the length of `lst`) and whose second component is the last element of `lst`. It returns `None` if `lst` is empty. For example,

   `split_last [1; 2; 3; 4]` returns `Some ([1; 2; 3], 4)`