Section A

Attempted questions: 2

Attached question: 2

a.

```
type nested_list =
    | Atom of int
    | Nest of nested_list list;;

let x = Nest [Nest [Atom 3;Atom 4];Atom 5;Nest [Atom 6;Nest [Atom 7];Atom 8];Nest []];;
```

b.

```
let rec flatten = function
    | Atom x -> [x]
    | Nest [] -> []
    | Nest (a::bs) -> (flatten a) @ (flatten (Nest bs));;
```

c.

```
let rec nested_map f = function
    | Atom x -> Atom (f x)
    | Nest [] -> Nest []
    | Nest (a::bs) -> (match (nested_map f (Nest bs)) with | Nest cs -> Nest ((nested_map f a)::cs));;
```

d.

```
(int -> int) -> nested_list -> nested_list
```

e.

```
let pack_as xs n =
    let rec aux xs n =
        match (xs, n) with
        | (a::bs, Atom _) -> (Atom a, bs)
        | (xs, Nest []) -> (Nest [], xs)
        | (xs, Nest (a::bs)) ->
            let (x, ys) = aux xs a in
            let ((Nest p), ps) = aux ys (Nest bs) in
            (Nest (x::p), ps)
    in let (r, q) = aux xs n in r;;
```

f. The data type `nested_zlist` is a lazy equivalent of `nested_list` in that it is like a `nested_list` except that the elements are not calculated until they are needed, as they are frozen by unit-accepting functions.

g.

```
let rec nested_zlist_to_list = function
    | ZAtom x -> Atom x
    | ZNest x -> (match x() with
        | [] -> Nest []
        | a::bs -> (match (nested_zlist_to_list (ZNest (fun () -> bs))) with
            | Nest cs -> Nest((nested_zlist_to_list a)::cs)));;
```