Assume this implementation of fold:

(*) fold : (a -> b -> b) -> b -> a list -> b *
fun fold f base [] = base
  | fold f base (h::t) = f h (fold f base t)

(A) With your group, use fold to write a function split of type:

   α list -> α list * α list

such that if split l = (a, b), then half of l’s elements are in a and the other half in b (don’t worry about order). Hint: your accumulator for fold will need to carry extra information.

(B) With your group, use references to implement fact and fib:
(C) Which way does our version of fold associate? Use fold to implement another version of fold that associates the other way.

(D) Implement a module satisfying this signature:

```plaintext
signature Ref = sig
    type t
    val create : int -> t
    val set : t -> int -> unit
    val get : t -> int
end
```