16:21 Mon 20. Jun

artemis.ase.in.tum.de — Private

```
  | x::xs -> i (i (x,x) [x]) xs
```

2. Write tail recursive versions of the following functions (without changing their types). In addition to the definition from the lecture, all functions must use constant stack space ($\mathcal{O}(1)$ in the size of its input). In particular, all the helper functions used need to be tail-recursive and use constant stack space too! If you use a library function, check that the documentation (e.g. for **List**) marks it as tail-recursive, or when in doubt implement a tail-recursive version yourself!

a)

```
let rec fac n =
  if n = 0 then 1
  else n * fac (n - 1)
```

或…= let … in … function |…|…

a) let rec go acc x = if x=1 then acc
else go (acc*x) (x-1)
let rec fac n = go 1 n.

*用List.init 造随机数列

b)

```
let rec remove a = function
  | [] -> []
  | x::xs -> if x = a then remove a xs else x::remove a xs
```

用function 处理ls更简

b) let rec remove a ls
= let rec remove a ls []
-helper.

将对象写作 function 的 match with accum 处理对象

*用 let in 可以用外面的参数，或少写个参数！

let rec remove_help a ls stor
if ls = [] then stor [终止]
else match ls with x::xs
→ if x=a then
remove_help a xs 接后
else remove help a xs stor

c)

```
let rec partition p l = match l with
  | [] -> [],[]
  | x::xs ->
    let a,b = partition p xs in
    if p x then x::a,b else a,x::b
```

Note: While the tests try to tell you whether your implementation is tail-recursive they may not be 100% accurate. Check your implementation for tail-recursion yourself.

*Note: OCaml 4.14 adds an experimental "Tail Modulo Constructor" (TMC) transformation (**docs**) that allows some non-tail-recursive functions to become tail-recursive automatically. This needs to be activated explicitly with the annotation `[@tail_mod_cons]`; see the documentation for more. You should implement these functions **without** TMC, as there may be problems in the future where this feature is not allowed.*

int list = Cons (int * 函数)