Reihungen

```
structure Array :> ARRAY = struct
   type 'a array = 'a ref vector
   fun array (n,x) = Vector.tabulate(n, fn  => ref x)
   fun fromList xs = Vector.fromList (map ref xs)
   fun sub (v,i) = !(Vector.sub(v,i))
   fun length v = Vector.length v
   fun foldl f s v = Vector.foldl (fn (x,a) => f(!x,a)) s v
   fun foldr f s v = Vector.foldr (fn (x,a) => f(!x,a)) s v
   fun app p v = Vector.app (fn x => p(!x)) v
   fun update (v,i,x) = Vector.sub(v,i) := x
   fun modify f a = iterup 0 (length a - 1) ()
       (fn (i,_) => update(a,i,f(sub(a,i))))
end
```

“In-Place” Reversieren

```
fun reverse a = let
   fun swap i j =
     Array.update (a,i, #1(Array.sub(a,j),
       Array.update(a,j,Array.sub(a,i))))
   fun reverse' l r =
     if l>=r then ()
     else (swap l r; reverse' (l+1) (r-1))
in
   reverse' 0 (Array.length a -1)
end
```

Imperative Schlangen

```
structure Queue :> QUEUE = struct
   datatype 'a cell = D | E of 'a * 'a entry
   withtype 'a entry = 'a cell ref
   type 'a queue = 'a entry ref * 'a entry ref
   fun queue () = let
     val dummy = ref D
   in
     (ref dummy, ref dummy)
   end
   fun snoc (_,f) x = let
     val dummy = ref D
   in
     !f:=E(x,dummy) ; f := dummy
   end
   fun tail (h,_) = case !(!h) of
       D => raise Empty | E (_,n) => h:=n
   fun head (h,_) = case !(!h) of
       D => raise Empty | E (x,_) => x
   fun empty (h,f) = !h = !f
end
```