Lista 7

1. (* Z1 *)
```
module type QUEUE_FUN =
sig
  type 'a t
  exception Empty of string
  val empty: unit -> 'a t
  val enqueue: 'a * 'a t -> 'a t
  val dequeue: 'a t -> 'a t
  val first: 'a t -> 'a
  val isEmpty: 'a t -> bool
end;;

(* a) *)
module Queue_fun: QUEUE_FUN =
struct
    type 'a t = 'a list
    exception Empty of string

    let empty () = []
    let enqueue (v, lst) = lst@[v]
    let dequeue = function
        [] -> raise (Empty "dequeue")
      | _::xs -> xs;;
    let first = function
        [] -> raise (Empty "first")
      | x::_ -> x;;
    let isEmpty lst = lst=[] (* lst=[] to sprawdzenie rownosci *)
end;;

(* b) *)
module Queue_fun2: QUEUE_FUN =
struct
    type 'a t = 'a list * 'a list
    exception Empty of string

    let normalize = function
        ([], q) -> (List.rev q, [])
      | q -> q (* Jeżeli lista jest postaci normalnej, nic nie rób *)

    let empty () = ([], [])
    let enqueue (v,(x1, lst)) = normalize (x1, v::lst)
    let dequeue = function
        (_::xs, q) -> normalize (xs, q)
      | _ -> raise (Empty "dequeue")
    let first = function
        (x::_, _) -> x
      | _ -> raise (Empty "first")
    let isEmpty lst = lst=([], []) (* lst=([],[]) to sprawdzenie rownosci *)
end;;
```

2. (* Z2 *)

```ocaml
module type QUEUE_MUT =
sig
  type 'a t
        (* The type of queues containing elements of type ['a]. *)
  exception Empty of string
        (* Raised when [first q] is applied to an empty queue [q]. *)
  exception Full of string
        (* Raised when [enqueue(x,q)] is applied to a full queue [q]. *)
  val empty: int -> 'a t
        (* [empty n] returns a new queue of length [n], initially empty. *)
  val enqueue: 'a * 'a t -> unit
      (* [enqueue (x,q)] adds the element [x] at the end of a queue [q]. *)
  val dequeue: 'a t -> unit
        (* [dequeue q] removes the first element in queue [q] *)
  val first: 'a t -> 'a
       (* [first q] returns the first element in queue [q] without removing
           it from the queue, or raises [Empty] if the queue is empty. *)
  val isEmpty: 'a t -> bool
        (* [isEmpty q] returns [true] if queue [q] is empty,
           otherwise returns [false]. *)
  val isFull: 'a t -> bool
        (* [isFull q] returns [true] if queue [q] is full,
           otherwise returns [false]. *)
end;;

module Queue_mut: QUEUE_MUT =
struct
    type 'a t = {mutable f:int; mutable r:int; mutable a: 'a option array }
    exception Empty of string
    exception Full of string

    let empty n = { f=0; r=0; a=Array.create (n+1) None }
    let enqueue (el, arr) =
      if (arr.r+1) mod (Array.length arr.a)=arr.f then raise (Full "enqueue")
                          else begin
                                 arr.a.(arr.r) <- Some el;
                                 arr.r <- (arr.r+1) mod (Array.length arr.a)
                          end
    let dequeue arr =
           if arr.f!=arr.r then arr.f <- (arr.f+1) mod (Array.length arr.a)
    let first arr =
                   if arr.f=arr.r then raise (Empty "first")
                   else match arr.a.(arr.f) with
                          Some e -> e
                          | None -> failwith "To i tak sie nigdy nie zdarzy"
    let isEmpty arr = arr.f=arr.r
    let isFull arr = (arr.r+1) mod (Array.length arr.a)=arr.f
end;;
```