宣言型プログラム論 課題6

200911434 青木大祐

平成 24 年 10 月 23 日

6-1

```
module type QUEUE =
sig
  type 'a t
  val empty: 'a t
  val enq : 'a t -> 'a -> 'a t
  val null : 'a t -> bool
  val deq : 'a t -> 'a * 'a t
end;;

module Queue3:QUEUE =
struct
  type 'a t = 'a list * 'a list
  let empty = ([], [])
  let norm (l1, l2) =
    match l1 with
        [] -> (List.rev l2, [])
      | _ -> (l1, l2)
  let enq (l1, l2) x = norm (l1, x::l2)
  let null q =
    match q with
        ([],[]) -> true
      | _  -> false
  let deq (l1,l2) =
    match l1 with
        x::l1' -> (x, norm (l1', l2))
      | _ -> raise (Failure "deq")
end

(* 6.1.1 *)
let enqlist q list =
  List.fold_left (fun queue x -> Queue3.enq queue x) q list
;;

(* 6.1.2 *)
let print_int_queue q =
  let tmp = ref q in
  try
    while true do
      let x,queue = Queue3.deq !tmp in
      tmp := queue;
      print_int x;
      print_string "\n"
    done
  with Failure "deq" -> ()
;;

let q = enqlist Queue3.empty [1;2;3];;
let x,q = Queue3.deq q;;
let x,q = Queue3.deq q;;
let x,q = Queue3.deq q;;

let q = enqlist Queue3.empty [1;2;3];;
print_int_queue q;;
```

実行結果

```
# val enqlist : 'a Queue3.t -> 'a list -> 'a Queue3.t = <fun>
# val print_int_queue : int Queue3.t -> unit = <fun>
# val q : int Queue3.t = <abstr>
# val x : int = 1
# val q : int Queue3.t = <abstr>
# val x : int = 2
# val q : int Queue3.t = <abstr>
# val x : int = 3
# val q : int Queue3.t = <abstr>
# val q : int Queue3.t = <abstr>
# 1
2
3
- : unit = ()
```

6-2

```
(* question 6.2 *)
module type QUEUE =
sig
```

```
  type 'a t
  val empty: 'a t
  val enq : 'a t -> 'a -> 'a t
  val null : 'a t -> bool
  val deq : 'a t -> 'a * 'a t
end;;

module Queue3:QUEUE =
struct
  type 'a t = 'a list * 'a list
  let empty = ([], [])
  let norm (l1, l2) =
    match l1 with
        [] -> (List.rev l2, [])
      | _ -> (l1, l2)
  let enq (l1, l2) x = norm (l1, x::l2)
  let null q =
    match q with
        ([],[]) -> true
      | _  -> false
  let deq (l1,l2) =
    match l1 with
        x::l1' -> (x, norm (l1', l2))
      | _ -> raise (Failure "deq")
end

let enqlist q list =
  List.fold_left (fun queue x -> Queue3.enq queue x) q list
;;

let print_int_queue q =
  let tmp = ref q in
  try
    while true do
      let x,queue = Queue3.deq !tmp in
      tmp := queue;
      print_int x;
      print_string "\n"
    done
  with Failure "deq" -> ()
;;

let graph1 = [("a", "b"); ("a", "c"); ("a", "d");
              ("b", "e"); ("c", "f"); ("d", "e");
              ("e", "f"); ("e", "g"); ("f", "d")
             ]
;;

let nexts v xs =
  List.fold_right (fun x l ->
    let (a, b) = x in
    if a = v then b::l else l) xs []
;;

(* 6.2.1 *)
nexts "a" graph1;;

let mem x ys = List.exists (fun y -> y = x) ys;;

let search graph v =
  let rec bfs q vs =  (* bfs: breadth first search *)
    if Queue3.null q then vs
    else
      let (v, q') = Queue3.deq q in
      if mem v vs then bfs q' vs
      else
        let vs = v::vs in
        let q = enqlist q' (nexts v graph) in
        bfs q vs
  in
  bfs (Queue3.enq Queue3.empty v) []
;;

(* 6.6.2 *)
search graph1 "a";;
```

実行結果

```
# val enqlist : 'a Queue3.t -> 'a list -> 'a Queue3.t = <fun>
# val print_int_queue : int Queue3.t -> unit = <fun>
# val graph1 : (string * string) list =
# [("a", "b"); ("a", "c"); ("a", "d"); ("b", "e"); ("c", "f"); ("d", "e");
   ("e", "f"); ("e", "g"); ("f", "d")]
# val nexts : 'a -> ('a * 'b) list -> 'b list = <fun>
# - : string list = ["b"; "c"; "d"]
```

```
# val mem : 'a -> 'a list -> bool = <fun>
# val search : ('a * 'a) list -> 'a -> 'a list = <fun>
# - : string list = ["g"; "f"; "e"; "d"; "c"; "b"; "a"]
```