```
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;;

module Queue : QUEUE_FUN =
struct
 type 'a t = 'a list
 exception Empty of string
  let empty() = []
 let enqueue (e, q) = match q with
                        | [] -> [e]
                        | l -> l@[e]
 let dequeue  = function
                | hd::tl -> tl
                | [] -> []
 let first = function
                | hd::tl -> hd
                |[] -> raise (Empty "empty queue")
 let isEmpty q = q = []

end ;;

let qq = Queue.empty ();;

let qq1 = Queue.enqueue (1, qq) ;;
let qq2 = Queue.enqueue (2, qq1) ;;
let qq3 = Queue.enqueue (3, qq2) ;;

Queue.first qq1 ;;
let qq22 = Queue.dequeue qq2;;
Queue.first qq33;;
Queue.first qq2;;
```

```
Queue.dequeue qq;;
Queue.first qq22;;

Queue.isEmpty qq;;


module DQueue : QUEUE_FUN =
struct
 type 'a t = 'a list * 'a list
 exception Empty of string
  let empty() = [],[]
 let enqueue (e, q) = match q with
                        | [],[] -> [e],[]
                        | [],l2 -> List.rev (e::l2), []
                        | l1, l2 -> l1, e::l2
 let dequeue  = function
                | [],[] -> [],[]
                | [], l2 -> let hd::tl = (List.rev l2) in tl,[]
                | [felem],l2 -> (List.rev l2), []
                | hd::tl,l2 -> tl,l2
 let first = function
                | hd::tl,l2 -> hd
                |[],[] -> raise (Empty "empty queue")
 let isEmpty q = q = ([],[])
end ;;



let qq = DQueue.empty ();;

let qq1 = DQueue.enqueue (1, qq) ;;
let qq2 = DQueue.enqueue (2, qq1) ;;
let qq3 = DQueue.enqueue (3, qq2) ;;

DQueue.first qq1 ;;
let qq22 = DQueue.dequeue qq2;;
DQueue.first qq33;;
DQueue.first qq2;;
DQueue.dequeue qq;;
DQueue.first qq22;;
```

```
DQueue.isEmpty qq;;
```

```
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 CArrQueue : QUEUE_MUT =
struct

 type 'a t = {mutable size: int; mutable f: int; mutable r: int;
mutable arr: 'a option array}
 exception Empty of string
 exception Full of string

 let empty s = {size = s +1; f = 0; r = 0; arr = Array.make (s+1) None}

 let isEmpty q = q.r = q.f
 let isFull q = (q.r +1)mod q.size = q.f

 let enqueue (elem, q) = if isFull q then raise (Full "full queue")
                      else (Array.set q.arr q.r (Some elem); q.r <-
((q.r +1)mod q.size);)
  let dequeue q = if isEmpty q then ()
                  else (Array.set q.arr q.f None; q.f <- ((q.f +1)mod
q.size);)

 let first q = if isEmpty q then raise (Empty "empty queue")
               else let Some temp = Array.get q.arr q.f in temp
end ;;

let cq1 = CArrQueue.empty 4;;

CArrQueue.isFull cq1;;
CArrQueue.isEmpty cq1;;

CArrQueue.enqueue (1,cq1);;

CArrQueue.isEmpty cq1;;

CArrQueue.enqueue (2,cq1);;

CArrQueue.enqueue (3,cq1);;

CArrQueue.enqueue (4,cq1);;

CArrQueue.isFull cq1;;
```

```
CArrQueue.first cq1;;

CArrQueue.dequeue cq1;;

CArrQueue.first cq1;;

CArrQueue.dequeue cq1;;

CArrQueue.first cq1;;

CArrQueue.dequeue cq1;;

CArrQueue.first cq1;;

CArrQueue.dequeue cq1;;

CArrQueue.first cq1;;

CArrQueue.isEmpty cq1;;


# CArrQueue.isFull cq1;;
- : bool = false
# CArrQueue.isEmpty cq1;;
- : bool = true
# CArrQueue.enqueue (1,cq1);;
- : unit = ()
# CArrQueue.isEmpty cq1;;
- : bool = false
# CArrQueue.enqueue (2,cq1);;
- : unit = ()
# CArrQueue.enqueue (3,cq1);;
- : unit = ()
# CArrQueue.enqueue (4,cq1);;
- : unit = ()
# CArrQueue.isFull cq1;;
- : bool = true
# CArrQueue.first cq1;;
- : int = 1
```

```
# CArrQueue.dequeue cq1;;
- : unit = ()
# CArrQueue.first cq1;;
- : int = 2
# CArrQueue.dequeue cq1;;
- : unit = ()
# CArrQueue.first cq1;;
- : int = 3
# CArrQueue.dequeue cq1;;
- : unit = ()
# CArrQueue.first cq1;;
- : int = 4
# CArrQueue.isEmpty cq1;;
- : bool = false
# CArrQueue.dequeue cq1;;
- : unit = ()
# CArrQueue.first cq1;;
Exception: CArrQueue.Empty "empty queue".
# CArrQueue.isEmpty cq1;;
- : bool = true
```

```
module Queue : QUEUE_FUN =
struct
 type 'a t = { mutable l: 'a list }
 exception Empty of string
  let empty() = { l = [] }
 let enqueue (e, q) = q.l <- e::q.l; q
 let dequeue q = match List.rev q.l with
                 | hd::tl -> q.l <- List.rev tl; q
                 |[] -> q
 let first q = match List.rev q.l with
               | hd::tl -> hd
               |[] -> raise (Empty "empty queue")
 let isEmpty q = q.l = []

end ;;
```