```
(* ESERCIZIO: SOLUZIONE *)

exception Undefined;;

module type RELAZIONE =
 sig type ('a,'b) relazione
  val relazioneC: unit -> ('a,'b) relazione
  val isIn1: ('a,'b) relazione -> 'a -> bool
  val getAll2: ('a,'b) relazione -> 'a -> 'b list
  val add: ('a,'b) relazione -> 'a -> 'b -> ('a,'b) relazione
end;;


module Relazione =
 (struct
  type ('a,'b) relazione = ('a * 'b) list
  let relazioneC = fun () -> []
  let rec isIn1 r x = match r with
            |[] -> false
            |(u,v)::rs -> (u=x) || isIn1 rs x
  let rec getAll2 r x = match r with
            |[] -> []
            |(u,v)::rs -> let s = (getAll2 rs x)
                          in if (u=x) then v::s else s
  let add r x y = (x,y)::r
end:RELAZIONE);;

(* short test *)
let valore = Relazione.relazioneC();;
let valoreA = Relazione.add valore "A" 5;;
let x = Relazione.isIn1 valore "A";;
Relazione.isIn1 (Relazione.add valoreA "B" 7) "A";;


(* Esercizio 1.
Completare usando la rappresentazione indicata sotto:
*)
module Relazione1 =
 (struct
  type ('a,'b) relazione =  ('b list) * (('a * 'b) -> bool)
  let relazioneC = fun () -> let f = fun x -> false
                             in ([],f)
  let isIn1 = fun r x -> let (d2,f) = r
                         in let rec h = fun d -> match d with
                                                 |[] -> false
                                                 |y::ds -> f(x,y) || h ds
                            in h d2
  let getAll2 = fun r x -> let (d2,f) = r
                           in let rec h = fun d -> match d with
                                                   |[] -> []
                                                   |y::ds -> let g = h ds
                                                             in if f(x,y) then y::g else g
                            in h d2
  let add = fun r u v  -> let (d2,f) = r
                          in let g = fun x -> if x=(u,v) then true else f x
                             in (v::d2,g)
end:RELAZIONE);;

(* Esercizio 2.
Provvedere ad uno short test per Relazione1, caricare ed eseguire.
```

```
*)

(* Esercizio 3.
Definire un nuovo modulo Relazione2 che riscriva le definizioni di Relazione1, utilizzando
List.fold_right e List.map, in modo appropriato ed eliminando l'uso definizioni ricorsive (
module Relazione2 =
 (struct
  type ('a,'b) relazione =  ('b list) * (('a * 'b) -> bool)
  let foldR = List.fold_right;;
  let map = List.map;;
  let app = List.append;;
  let relazioneC = fun () -> let f = fun x -> false in ([],f)
  let isIn1 (d2,f) x = foldR (||) (map(fun y -> f(x,y)) d2) false
  let getAll2 (d2,f) x = foldR (app) (map(fun y -> if f(x,y) then [y] else [])d2) []
  let add (d2,f) u v = let g = fun x -> if x==(u,v) then true else f x
                       in (v::d2,g)
end:RELAZIONE);;
*)

(* Esercizio 4.
Estendere il tipo astratto ('a,'b) relazione in modo tale che in aggiunta alle operazioni g
ora disponibili le operazioni:
    - proj1, che restituisce la lista dei primi componenti della relazione;
    - proj2, che restituisce la lista dei secondi componenti della lista;
    - size, che restituisce la lista delle differenti coppie della relazione.
Si fornisca un'implementazioneper tale API. Si utilizzino List.fold_right e List.map.
*)
```