```
(* EXERCICE 1 *)
let sinc (x:float):float = match x with
  |0. -> 1.
  |_ -> (sin x)/.x;;

(* EXERCICE 2 *)
let f (x,y) z = x || y || z;;
(* val f : bool * bool -> bool -> bool = <fun> *)
let g a = a^".";;
(* val g : string -> string = <fun> *)
let h b v = if b then v else 0.;;
(* val h : bool -> float -> float = <fun>  *)

(* EXERCICE 3 *)
type reel_etendu = Reel of float | PlusInf | MoinsInf;;

let add_etendue (x:reel_etendu) (y:reel_etendu) :reel_etendu = match (x,y) with
  |(Reel x, Reel y) -> Reel (x +. y)
  |(Reel x, PlusInf) -> PlusInf
  |(PlusInf, Reel y) -> PlusInf
  |(Reel x, MoinsInf) -> MoinsInf
  |(MoinsInf, Reel y) -> MoinsInf
  |(PlusInf,PlusInf) -> PlusInf
  |(MoinsInf,MoinsInf) -> MoinsInf
  |(PlusInf,MoinsInf) -> failwith "erreur"
  |(MoinsInf,PlusInf) -> failwith "erreur";;

let add_etendue_concis (x:reel_etendu) (y:reel_etendu) :reel_etendu = match (x,y) with
  |(Reel x, Reel y) -> Reel (x +. y)
  |(Reel x, PlusInf) -> PlusInf
  |(PlusInf, Reel y) -> PlusInf
  |(Reel x, MoinsInf) -> MoinsInf
  |(MoinsInf, Reel y) -> MoinsInf
  |(a,b) when a=b -> a
  |_ -> failwith "erreur";;

(* PROBLÈME *)
type complexe = float*float;;

let re (z:complexe) :float = let (x,y) = z in x;;
let im ((x,y):complexe) :float = y;;

let conj ((x,y):complexe) :complexe = (x,-.y);;

let add ((a,b):complexe) ((c,d):complexe) :complexe = (a+.c,b+.d);;

let mul ((a,b):complexe) ((c,d):complexe) :complexe = (a*.c-.b*.d,a*.d+.b*.c);;

let norm ((x,y):complexe) :float = sqrt (x*.x +. y*.y);;

let arg ((x,y):complexe) :float = match (x,y) with (* on suppose z non nul *)
  |(x,y) when x < 0. -> 4.*.(atan 1.)
  |_ -> 2.*.(atan y/.(x+. (norm (x,y))));;
```