```
(*
 Si consideri la Sintassi Astratta sotto, per il Lambda Calcolo:
        Lambda::= Var | Const | [$] Var Lambda | [@] Lambda Lambda
 dove usiamo $ per l'astrazione e @ per l'applicazione.
 Si chiede di fornire in OCaml strutture, tipi, operazioni per la
 costruzione, l'accesso e la presentazione degli Abstract Syntax
 Tree dei termini del Linguaggio del Lambda Calcolo.
 In particolare, le operazioni richieste devono includere:
 mkConst, mkVar, mkAbs, mkApp,
 isConst, isVar, isAbs, isApp
 getIde, getTem, getFun, getArg
 printAST (per la presentazione degli AST)
 printCRT (per la presentazione nella sitassi concreta, vedi Lezioni,
           degli AST)
*)

(*
 Questo modulo e' il corrispettivo del modulo header della versione
 fornita in C, vedi modulo Last.h
*)
open Printf;;                (* un modulo di OCaml con operazioni per ... *)
#use "LastH.ml";;

(* operazioni di costruzione dei valori *)

let mkConst = fun x -> Const x;;

let mkVar x = Var x;;        (* un' altra forma per binding funzionali *)

let mkAbs x e = Abs(x,e);;

let mkApp e1 e2 = App(e1,e2);;


(* predicati di sottotipo *)
(* Come visitare, accedere, "modificare", valori e componenti di tipi
   algebrici.
   Pattern e Pattern Matching nei valori strutturati con costruttori:
   liste, tuple, tipi algebrici.
   I valori sono termini di costruttori. Ad es. 3::5::7::18 e' un termine t.
   I patterns sono termini di costruttori con variabili (libere) al posto di
   sotto-termini. Ad es. x::y::z e' un pattern p.
   Operazione pattern matching (nei Linguaggi di Programmazione Applicativi):
        match(t,p) = calcola ambiente rho, di binding per variabili di p che
                     rendono p identico a t (i.e. t = rho(p)), se esiste.
   Costrutto match in OCaml
        match t with p1 -> t1 | ... | pn -> tn
   calcola rho(ti) se esiste minimo i<=n, tale che t=rho(pi)
   Pattern Speciali:
        match(t,x) = [t/x], ambiente con binding (x,t)
        match(t,_) = [t/?], ambiente con binding per anonima variabile _
        match(c(t1,...,tk),c(p1,...pk)) = match(tk,rho1(...rhok-1(pk)...))
*)

let isConst x = match x with Const _ -> true | _ -> false;;

let isVar = function Var _ -> true | _ -> false;;
                             (* Sopra: Un'altra forma di binding funzionale. *)
                             (* Definizione per casi: L'arg. e' una var. anonima *)
let isAbs = function Abs _ -> true | _ -> false;;

let isApp = function App _ -> true | _ -> false;;


(* operazioni di selezione *)
exception Error;;

let getIde x = match x with
        | Const v -> v
        | Var v -> v
        | Abs(v,_) -> v
        | _ -> raise Error
```

```
        ;;
let getTerm = function
        | Abs(_,t) -> t
        | _ -> raise Error
        ;;
let getFun = function
        | App(t1,_) -> t1
        | _ -> raise Error
        ;;
let getArg = function
        | App(_,t2) -> t2
        | _ -> raise Error
        ;;

(* operazioni di presentazione *)
(* costrutto sprintf:  sprintf stringa_formato_k e1 ... ek *)

let rec printAST = function
        | Const v -> sprintf "[%s]" v
        | Var v -> sprintf "[%s]" v
        | Abs(x,t) -> sprintf "[$-([%s],%s)]" x (printAST t)
        | App(t1,t2)-> sprintf "[@-(%s,%s)]" (printAST t1)(printAST t1)
        ;;
```