Linguagens e Ambientes de Programação

Exame de Época de Recurso 2008/2009 – Proposta de Resolução

1.

A	B	C	D	E	F
c	c	d	c	a	b

A) No caso de uma linguagem com aninhamento de funções, a função passada como argumento pode ter necessidade de aceder a variáveis intermédias e por isso, juntamente com o endereço da função, também tem de ser passado o respectivo ambiente de definição (static link). (*aula teórica 14*)

E) Polimorfismo paramétrico é uma forma de polimorfismo universal onde a função polimórfica tem um parâmetro de tipo implícito ou explícito. Na chamada da função o parâmetro de tipo pode ser ou não inferido. (*aula teórica 26*)

2.
```
f : 'a list -> 'a -> ('a * bool) list
```

3.

a)
```
let rec courses g =
    match g with
        | [] -> []
        | (_,c,_)::xs ->
            let cs = courses xs in
                if List.mem c cs then cs
                else c::cs
;;
```

b)
```
let rec averages g =
    let c = courses g in
        averagesAux c g
;;

let rec averagesAux c g =
    match c with
        | [] -> []
        | x::xs ->
            let (gr,n) = sum x g in
                (x,(float_of_int gr /.
                    float_of_int n))::averagesAux xs g
;;

let rec sum c g =
    match g with
        | [] -> (0,0)
        | (_,c1,g1)::xs ->
            match sum c xs with
                | (0,0) ->
                    if c1 = c then (g1,1)
                    else (0,0)
                | (g2,n) ->
                    if c1 = c then (g1+g2, n+1)
                    else (g2,n)
;;
```

c)
```
let rec prec s t =
    match t with
        | Nil -> []
        | Node(c,pl,pr) ->
            if c = s then precAux pl @ precAux pr
            else prec s pl @ prec s pr
;;

let rec precAux t =
    match t with
        | Nil -> []
        | Node(c,pl,pr) -> c::(precAux pl @ precAux pr)
;;
```

d)
```
let rec valid t =
    checkRepetitions (notLeafs t) && checkLoops t []
;;

let rec notLeafs t =
    match t with
        | Nil -> []
        | Node(c,pl,pr) ->
            if pl != Nil || pr != Nil
                then c :: notLeafs pl @ notLeafs pr
            else notLeafs pl @ notLeafs pr
;;
```

```
let rec checkRepetitions l =
    match l with
        | [] -> true
        | x::xs ->
            if List.mem x xs then false
            else checkRepetitions xs
;;

let rec checkLoops t prev =
    match t with
        | Nil -> true
        | Node(c,pl,pr) ->
            if List.mem c prev then false
            else let prev = c::prev in
                checkLoops pl prev && checkLoops pr prev
;;
```

4.

25:	PC	?	-- A --
	SL	00	
	DL	19	
	i	936	-- C1 --
	PC	?	
20:	SL	16	
	DL	16	
	PC	?	-- B1 --
	SL	07	
	DL	13	
15:	PC	?	-- C2 --
	SL	10	
	DL	10	
	PC	?	-- B2 --
	SL	07	
10:	DL	07	
	PC	?	-- A --
	SL	00	
	DL	04	
	PC	?	-- main --
05:	SL	00	
	DL	00	
	pt	22	-- start --
	PC	?	
	SL	?	
00:	DL	?	