Linguagens e Ambientes de Programação

Exame de Época de Recurso 2007/2008 – Proposta de Resolução

1. **V** Em linguagens com tipificação estática, como é o caso do ML, C e C++, os erros de tipo são detectados em tempo de compilação, dado que os tipos estão associados às expressões que ocorrem no texto dos programas, e é o próprio texto que é validado.

   F Num programa validado, ou seja sem erros de tipo, podem ainda assim ocorrer diversos problemas durante a execução.

   V, **F**, **V**, **F**, **V**, **F**, **V**, **V**

2.
   1. A função `f` recebe um único argumento, uma lista de pares ordenados (por causa do padrão) e o seu resultado é um inteiro, dado que `f` devolve zero no caso da lista vazia.

      ```
      f : (__ * __) list -> int
      ```

   2. Para inferir o tipo das componentes de cada par ordenado, é necessário observar como essas componentes são utilizadas no corpo da função. De `a (b (f xs))` deduz-se que `a` e `b` são funções com um único argumento.

      ```
      f : ((__ -> __) * (__ -> __) ) list -> int
      ```

   3. A função `b` recebe o resultado de `f`, `int`, e retorna valores de um tipo desconhecido `'a`. A função `a` recebe o resultado de b, ‘a, e retorna int, o tipo do retorno de f.

   Assim sendo:

   ```
   f : (('a -> int) * (int -> 'a)) list -> int
   ```

3.

a)
```
let rec rightmost i t =
    match t with
        | Entity e -> if String.length e == i then e else ""
        | Question(q,n,y) -> let s = rightmost i y in
                                if s <> "" then s
                                else rightmost i n
;;
```

b)
```
let rec cmp t1 t2 =
    match t1, t2 with
        | Entity e1, Entity e2 ->
            Node((if e1 = e2 then 1 else 0), Nil, Nil)
        | Question(q1,n1,y1), Question(q2, n2, y2) ->
            Node((if q1 = q2 then 1 else 0), cmp n1 n2,
                                                cmp y1 y2)
        | Entity e, Question(q,n,y) ->
            Node(-1, cmpAux n, cmpAux y)
        | Question(q,n,y), Entity e ->
            Node(-1, cmpAux n, cmpAux y)
;;

let rec cmpAux t =
    match t with
        | Entity e -> Node(-2,Nil,Nil)
        | Question(q,n,y) -> Node(-2, cmpAux n, cmpAux n)
;;
```

c)
```
let rec dm s t =
    match t with
        | Entity e -> 1001
        | Question(q,n,y) ->
            let hasLeft = hasNode s n in
                let hasRight = hasNode s y in
                    if hasLeft && hasRight
                        then min (min (dm s n) (dm s y))
                            ((depth s n) + (depth s y))
                    else if hasLeft && not hasRight && q = s
                        then min (1 + depth s n) (dm s n)
                    else if hasRight && not hasLeft && q = s
                        then min (1 + depth s y) (dm s y)
                    else 1001
;;

let rec hasNode s t =
    match t with
        | Entity e -> if s = e then true
                      else false
        | Question(q,n,y) -> if s = q then true
                             else hasNode s n && hasNode s y
;;

let rec depth s t =
    match t with
        | Entity e -> 0
        | Question(q,n,y) -> if q = s then 0
                         else min (1 + depth s n) (1 + depth s y)
;;
```

4.

	v		-- F --
	PC	?	
	SL	00	
20:	DL	14	
	n	2	
	a	13	
	i	0 -> 1 -> 2 -> 3 -> 4	-- Update --
	PC	?	
15:	SL	10	
	DL	10	
	v	{ 91, 92, 93, 94 }	-- F --
	PC	?	
	SL	00	
10:	DL	04	
	n	1	
	a	3	
	x	1	-- main --
	PC	?	
05:	SL	00	
	DL	00	
	v	{ 90, 91, 92, 93 }	-- start --
	PC	?	
	SL	?	
00:	DL	?	

6.

```
int CompareTo(const void *a, const void *b) {
  List la = *((List *) a);
  List lb = *((List *) b);

  return la->data - lb->data;
}

List ListSort(List l) {
  List *v = malloc(sizeof(List) * Size(l));
  int size = Size(l);
  int i;
  List temp = l;

  for(i = 0; temp != NULL; temp = temp->next)
        v[i++] = temp;

  qsort(v, size, sizeof(List), CompareTo);

  for(i = 0; i < size-1; i++)
        v[i]->next = v[i + 1];

  v[size-1]->next = NULL;

  temp = v[0];
  free(v);
  return temp;
}
```