PROGRAMMAZIONE 1 e LABORATORIO (A,B) - a.a. 2014-2015
Prova scritta del 15/07/2015

SOLUZIONI PROPOSTE

Per ogni esercizio vengono proposte una o più soluzioni tra le tante possibili.

ESERCIZIO 1 (6 punti)
Scrivere una grammatica **regolare** che genera il seguente linguaggio sull'alfabeto $\Lambda = \{a, b, c, d\}$

$$\mathcal{L} = \{x\alpha x |\ x \in \Lambda \wedge \alpha \in \Lambda^*\}$$

Soluzione

```
S ::= aA | bB | cC | dD
A ::= aA | bA | cA | dA | a
B ::= aB | bB | cB | dB | b
C ::= aC | bC | cC | dC | c
D ::= aD | bD | cD | dD | d
```

ESERCIZIO 2 (6 punti)
Dato un albero binario, si definisce *livello* di un nodo nell'albero il numero di nodi che si incontrano nel cammino dalla radice al nodo medesimo. Quindi, ad esempio, la radice ha livello 1, i figli della radice hanno livello 2, e così via. Dato il tipo degli alberi binari visti a lezione

```
type 'a btree = Void | Node of 'a * 'a btree * 'a btree
```

si scriva in CAML una funzione

```
foo : 'a btree -> int -> 'a list * 'a list
```

che, dato un albero binario e un intero positivo n, restituisce una coppia di liste $(l1, l2)$ in cui $l1$ contiene tutti e soli i valori dei nodi che occorrono a livelli dell'albero minori o uguali a n e $l2$ contiene tutti e soli i valori degli altri nodi dell'albero.

Soluzione

```
let rec foo bt n = match bt, n with
     Void, _ -> [],[]  |
     Node(x, lt, rt), 0 ->
            let _, l1 = foo lt 0 in
               let _, l2 = foo rt 0
                   in [], x::(l1@l2)  |
     Node(x, lt, rt), m when m > 0 ->
            let l1, l2 = foo lt (m-1) in
               let l3, l4 = foo rt (m-1)
                   in x::(l1@l3), l2@l4;;
```

ESERCIZIO 3 (6 punti)
Si definisca in CAML, senza utilizzare ricorsione esplicita, una funzione

```
twice : 'a -> 'a list -> bool
```

in modo che `(twice x xs)` restituisca `true` se `x` occorre in `xs` esattamente due volte, restituisca `false` altrimenti.

Soluzione

```
let twice x xs = let f z n = if z=x then n+1 else n in (foldr f 0 xs)=2;;
```

ESERCIZIO 4 (6 punti)
Si scriva in `C` una procedura che, prese attraverso opportuni parametri due liste di interi, elimina dalla seconda lista tutti gli elementi che non occorrono anche nella prima.
Si suppongano predefiniti i tipi

```
struct el {int info; struct el *next;};

typedef struct el ElementoDiLista;
typedef ElementoDiLista *ListaDiInteri;
```

Soluzione

```
int member (ListaDiInteri l, int x)
{
   int trovato = 0;
   while (l != NULL && !trovato)
     if (l->info == x)
       trovato = 1;
     else
       l = l->next;
   return trovato;
}

void del (ListaDiInteri prima, *seconda)
{
   ListaDiInteri prec=NULL, corr=*seconda;
   while (corr != NULL)
     {
       if (!member(prima, corr->info))
          {
            if (prec==NULL)
               { *seconda = *seconda -> next;
                 free(corr);
                 corr = *seconda;
               }
            else
               {
                 prec->next = corr->next;
                 free(corr);
                 corr = prec->next;
               }
          }
       else
          {
            prec = corr;
            corr = corr->next;
          }
     }
}
```

ESERCIZIO 5 (6 punti)
Si scriva in `C` una funzione che, dati due array di interi `a` e `b` e le loro dimensioni `dima` e `dimb`, restituisce il valore di verità della seguente formula

$$\exists i \in [0, dima).\#\{j \mid j \in [0, dimb) \ \wedge \ a[i] = b[j]\} > \#\{k \mid k \in [0, dima) \ \wedge \ a[i] = a[k]\}$$

Si ricorda che dato un insieme finito A, $\#A$ indica il numero di elementi di A.

Soluzione

```
int check (int a[], int b[], int dima, int dimb)
{
   int i=0, trovato = 0;
   while (i < dima & !trovato)
     {
       int conta_a = 0;
       int conta_b = 0;
       int j=0, k=0;

       for (j=0, j<dimb, j++)
         if (a[i]==b[j]) conta_b=conta_b+1;

       for (k=0, k<dima, k++)
         if (a[i]==a[k]) conta_a=conta_a+1;

       trovato = conta_b > conta_a;
       i = i+1;
     }
   return trovato;
}
```