PROGRAMMAZIONE 1 e LABORATORIO (A,B) - a.a. 2014-2015
Prova scritta del 1/09/2015

SOLUZIONI PROPOSTE

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

ESERCIZIO 1 (6 punti)
Si dimostri, utilizzando il pumping lemma, che il seguente linguaggio sull'alfabeto $\{a, b\}$ non è regolare

$$\mathcal{L} = \{ab^n ab^{n+1} \mid n > 0\}$$

Soluzione
Per dimostrare che il linguaggio non è regolare utilizzando il pumping lemma mostriamo quanto segue:

$$\forall n \in \mathbb{N}.\ \exists w \in \mathcal{L}.\ \mid w \mid \geq n \wedge \forall x, y, z. \neg\big(w = xyz\ \wedge\ y \neq \epsilon \wedge\ \mid xy \mid \leq n\ \wedge\ (\forall i \in \mathbb{N}. xy^i z \in \mathcal{L})\big)$$

ovvero

$$\forall n \in \mathbb{N}.\ \exists w \in \mathcal{L}.\ \mid w \mid \geq n \wedge \forall x, y, z. \big(w = xyz\ \wedge\ y \neq \epsilon \wedge\ \mid xy \mid \leq n) \Rightarrow\ \neg(\forall i \in \mathbb{N}. xy^i z \in \mathcal{L})\big)$$

Sia allora $n \in \mathbb{N}$ e consideriamo, tra le tante possibili, la stringa

$$w = ab^n ab^{n+1}$$

Chiaramente $w \in \mathcal{L}$ e $\mid w \mid \geq n$. Siano dunque x, y, z tali che:

(a) $w = xyz$

(b) $y \neq \epsilon$

(c) $\mid xy \mid \leq n$

Ragioniamo per casi:
(1) $x = \epsilon\ \wedge\ y = ab^h$, con $0 \leq h \leq n-1$ (e in questo caso $z = b^{(n-h)}ab^{n+1}$).
La stringa $xy^0z = z = b^{(n-h)}ab^{n+1} \notin \mathcal{L}$ c.v.d.

(2) $x = ab^h\ \wedge\ y = b^t$, con $t > 0$ e $h + t \leq n - 1$ (e in questo caso $z = b^{(n-h-t)}ab^{n+1}$).
La stringa $xy^0z = ab^h b^{(n-h-t)}ab^{n+1} = ab^{(n-t)}ab^{n+1} \notin \mathcal{L}$ poiché $t > 0 \implies n - t < n$ c.v.d.

ESERCIZIO 2 (6 punti)
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 -> 'a list * 'a list
```

che, dato un albero binario restituisce una coppia di liste $(l1, l2)$ in cui $l1$ contiene tutti i valori dei nodi dell'albero con esattamente due figli e $l2$ tutti i valori degli altri nodi dell'albero. Ad esempio

```
foo(Node(1, Node(2, Void, Void), Node(4, Node(3, Void, Void), Node(1,Void,Void)))) = [1;4], [2;3;1]
```

Soluzione

```
let rec foo bt = match bt with
     Void -> [],[]  |
     Node(x, lt, rt) ->
            let l1,l2 = foo lt in
               let l3, l4 = foo rt
                   in
                       if (lt=Void or rt=Void)
                           then l1@l3, x::(l2@l4)
                           else x::(l1@l3), (l2@l4);;
```

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

```
foo : ('a -> bool) -> 'a list -> 'a list
```

in modo che `(foo p xs)` restituisca una lista che contiene tutti gli elementi di `xs` e in cui gli elementi che soddisfano `p` precedono gli elementi che non soddisfano `p`.

Soluzione
Proponiamo due soluzioni.

```
let foo p xs =
    let q x = not (p x) in (filter p xs) @ (filter q xs);;
```

```
let foo p xs = let f x y = if (p x) then x::y else y@[x] in (foldr f [] xs);;
```

ESERCIZIO 4 (6 punti)
Si scriva in C una procedura che, presi attraverso opportuni parametri una lista di interi e un intero x, aggiunge un elemento che contiene x prima di ogni elemento negativo presente nella lista.
Si suppongano predefiniti i tipi

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

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

Soluzione

```
void ins (ListaDiInteri *l, int x)
{
   if (*l != NULL)
   {
      ListaDiInteri corr, prec, new;

      prec = *l;
      corr = prec -> next;

      if (*l -> info < 0)
        {
           new = malloc(sizeof(ElementoDiLista));
           new -> info = x;
           new -> next = *l;
           *l = new;
        }

      while (corr != NULL)
        {
         if (corr -> info <0)
          {
           new = malloc(sizeof(ElementoDiLista));
           new -> info = x;
           new -> next = corr;
           prec -> next = new;
          }
         prec = corr;
         corr = corr -> next;
        }
   }
}
```

ESERCIZIO 5 (6 punti)
Si scriva in `C` una funzione che, dato un array di interi `a`, la sua dimensione `dima` e un intero `R`, restituisce il valore di verità della seguente formula

$$(\forall i, j \in [0, dima).\ j > i\ \wedge\ a[i] = a[j]\ \Longrightarrow\ j - i = R)$$

Soluzione

```
int check (int a[], int dima, int R)
{
   int i=0, trovato = 0;
   while (i < dima & !trovato)
     {
       int j=i+1;
       while (j<dima & !trovato)
         if (a[i]==a[j] & (j-i)!=R)
            trovato = 1;
         else
            j=j+1;
       i=i+1;
     }
   return (!trovato);
}
```