PROGRAMMAZIONE 1 e LABORATORIO (A,B) - a.a. 2011-2012
Prova scritta del 25/06/2012

SOLUZIONI PROPOSTE

Negli esercizi di programmazione in `C` si assuma predefinito il tipo `typedef enum {false, true} boolean`.

ESERCIZIO 1 (punti 6)
Il tipo `Cicoria di 'a` è definito come segue:

- La cicoria vuota ha tipo `Cicoria di 'a`
- Una foglia di cicoria (rappresentata da una lista di `'a`) e una `Cicoria di 'a` hanno tipo `Cicoria di 'a`

Il tipo CAML e' dato da

```
type 'a chicory = EmptyChic | Chic of 'a list * 'a chicory;;
```

Si scriva una funzione CAML che prende una cicoria di interi `c` e un intero `x` e restituisce true se `x` compare in tutte le foglie di cicoria di `c`.

Soluzione

```
let rec check_cic c x =
    let rec member l y = match l with
            [] -> false |
            z :: zs -> if y=z then member zs y else false
    in
      match c with
         EmptyChic -> true |
         Chic(l, c1) -> if member l x then check_cic c1 x else false;;
```

ESERCIZIO 2 (punti 6)
Sia definito il predicato *be* nel modo seguente
$be(a, b, j, dim) \equiv j \in [0, dim) \ \wedge \ (\forall i \in [0, j).\ a[i] = b[dim - i - 1])$
Si scriva una funzione `C`

```
int be_index(int a[], int b[], int dim)
```

dove `a` e `b` sono array di interi di dimensione `dim` che restituisce

- il valore -1 se $\nexists j.\ j \in [0, dim) \wedge be(a, b, j, dim)$
- il massimo valore dell'insieme

  $\{i \mid i \in [0, dim) \ \wedge \ be(a, b, i, dim)\}$

  altrimenti

Suggerimento: si cerchi di comprendere il significato della proprietà espressa dal predicato *be*.

Soluzione

```
int be_index(int a[], int b[], int dim)
{
  if a[0] != b[dim-1] return -1;
  else
     {
       int i=1;
       boolean diversi = false;
       while (i < dim && !diversi)
         if (a[i]==b[dim-i-1]) i++; else diversi = 1;
       return i;
     }
}
```

ESERCIZIO 3 (punti 6)
Scrivere una funzione `C`

```
int subseq (int a[], int b[], int dima, int dimb)
```

dove `dima` e `dimb` sono le dimensioni degli array `a` e `b` rispettivamente, che calcoli il numero di volte in cui l'array `a` compare come sottosequenza dell'array `b`.

Soluzione

```
int subseq (int a[], int b[], int dima, int dimb)
{
  int i=0, conta_sub=0;
  while (i<= dimb-dima)
    {
      int ia=0;
      int ib=i;
      boolean uguali = true;
      while (ia < dima && uguali)
        if (a[ia]==b[ib]) {ia++; ib++;} else uguali=false;
      if (uguali) conta_sub++;
      i++;
    }
}
```

ESERCIZIO 4 (punti 6)
Si completi la seguente definizione

```
 let split  l = let f x y = ...  in foldr f ... l
```

con tipo

```
 split  : 'a list -> 'a list * 'a list
```

in modo che `(split lista)` restituisca la coppia `(l1, l2)` tale che `l1@l2=lista`, gli elementi di `l2` siano tutti uguali tra loro e l'ultimo elemento di `l1`, se esiste, è diverso dagli elementi in `l2`

Soluzione

```
let split l = let f x (l1, l2) = match l2 with
                    [] -> ([], [x])
                    y::ys -> if x=y & l1=[] then (l1, x::l2) else (x::l1, l2)
          in  foldr f ([], []) l;;
```

ESERCIZIO 5 (punti 6)
Date le seguenti definizioni:

```
struct el {int info; struct el *next;};
typedef struct el ElementoDiLista;
typedef ElementoDiLista *ListaDiElementi;
```

scrivere in C una procedura che, dati in ingresso attraverso opportuni parametri un intero x e una lista di interi, sposta in ultima posizione il primo elemento della lista, se esiste, strettamente maggiore di x.

Soluzione
Si propongono due soluzioni.

Prima soluzione

```
void move (ListaDiElementi *lista, int x)
{
  if (*l != NULL)
    if (*l -> next != NULL)
      {
        if (*l -> info > x)
          {
            ListaDiElementi aux = *l -> next;
            while (aux -> next != NULL)
               aux = aux -> next;
            aux -> next = *l;
            *l = *l->next;
            aux->next->next = NULL;
          }
        else
          {
            ListaDiElementi toMove = NULL;
            ListaDiElementi prec = *l;
            ListaDiElementi corr = *l->next;
            boolean primo = true;
            while (corr != NULL)
              {
                if (corr->info > x && primo)
                   {
                     toMove = prec;
                     primo = false;
                   }
                 prec = corr;
                 corr = corr-> next;
              }
            if (!primo)
              {
                prec->next = toMove->next;
                toMove->next = toMove->next->next;
                prec->next->next=NULL;
              }
          }
      }
}
```

Seconda soluzione

```
void move (ListaDiElementi *l, int x)
{
  if (*l != NULL)
    if (*l -> next != NULL)
      {
        boolean trovato = false;
        ListaDiElementi corr=*l;
        ListaDiElementi prec=NULL, toMove=NULL;
        while (corr != NULL)
          {
           if (!trovato && corr->info > x)
               {
                 trovato = true;
                 toMove = prec;
               }
           prec=corr;
           corr=corr->next;
          }
         if (trovato)
           {
             if (toMove==NULL)
               {
                 prec->next=*l;
                 *l = *l -> next;
                 prec->next->next=NULL;
               }
             else
               {
                  prec->next=toMove->next;
                  toMove->next=toMove->next->next;
                  prec->next->next=NULL;

               }
           }
      }
}
```