PROGRAMMAZIONE 1 e LABORATORIO (A,B) - a.a. 2011-2012
Prova scritta del 16/07/2012

SOLUZIONI PROPOSTE

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

ESERCIZIO 1 (punti 6)
Dato il tipo degli alberi binari visto a lezione

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

si definisca in CAML una funzione che, dato un albero binario `bt` e un'etichetta `x` restituisce l'albero ottenuto da `bt` eliminando tutti i sottoalberi con radice etichettata da `x`.

Soluzione

```
let rec delete bt x = match bt with
   Void -> Void |
   Node(y, lt, rt) when y=x -> Void |
   Node(y, lt, rt) when y<>x -> Node(y, delete lt x, delete rt x)
```

ESERCIZIO 2 (punti 6)
Un array di interi `a` di dimensione `dim` si dice *quasi ordinato* se e solo se

$$\exists j \in [0, dim).\ (\forall i \in [0, j).a[i] < a[i+1])\ \wedge\ (\forall i \in (j, dim-1).a[i] < a[i+1])$$

Si scriva una funzione `C`

```
boolean qsorted(int a[], int dim)
```

che restituisce `true` se l'array `a` di dimensione `dim` è quasi ordinato, e restituisce `false` altrimenti.

Soluzione

```
boolean qsorted(int a[], int dim)
{
  int j=0;
  boolean trovato = false;
  while (j<dim-1 && !trovato)
    if (a[j]<a[j+1])
       j=j+1;
    else trovato = true;
  if (trovato)
     {
       trovato = false;
       j=j+1;
       while (j<dim-1 && !trovato)
       if (a[j]<a[j+1])
          j=j+1;
       else trovato = true;
     }
  return (!trovato);
}
```

ESERCIZIO 3 (punti 6)
Si definisca una funzione `C` che dati due array `a`, di dimensione `dima`, e `b`, di dimensione `dimb`, verifica che tutti gli elementi di `a` che occorrono in `b` occorrano in `b` almeno due volte.

Soluzione

```
boolean foo(int a[], int dima, int b[], int dimb)
{
  int ia, ib, conta;
  ia=0;
  boolean trovato = false;
  while (ia<dima & !trovato)
  {
    ib=0;
    conta=0;
    while (ib<dimb && conta<2)
      {
        if (a[ia]==b[ib])
          conta=conta+1;
        ib=ib+1;
      }
    if (conta==1)
      trovato = true;
    else
      ia=ia+1;
  }
return (!trovato);
}
```

ESERCIZIO 4 (punti 6)
Data una lista di interi ℓ ed un intero x, indichiamo con $x \in \ell$ il fatto che x occorre in ℓ. Si completi la seguente definizione

```
let check  lista z = let f x y = ...  in foldr f ... l
```

con tipo

```
check  : int list -> int -> bool * bool
```

in modo che `(check lista z)` restituisca la coppia `(b1, b2)` tale che

$\texttt{b1} \equiv (\forall n.n \in \texttt{lista} \Rightarrow n \geq \texttt{z})$
$\texttt{b2} \equiv (\exists n.n \in \texttt{lista} \wedge n \neq \texttt{z})$

Soluzione

```
let check lista z = let f x (b1, b2) = ((b1 & x>=z), (b2 or x<>z))
           in  foldr f (true, false) lista;;
```

ESERCIZIO 5 (punti 6)
Siano date le seguenti definizioni

```
struct el {char codice; int quantita; struct el *next;};
typedef struct el Prodotto;
typedef Magazzino *Prodotto;
```

Un lista di tipo `Magazzino` rappresenta i prodotti presenti in un magazzino: ogni tipo di prodotto è rappresentato da un carattere, il suo codice univoco, e da un intero che ne indica la rimanenza. Si scriva una procedura `C` che, dati in ingresso attraverso opportuni parametri un magazzino `m` e un tipo di prodotto `c`, aggiorna il magazzino a seguito della vendita di una unità di quel prodotto.
N.B.: il tipo di prodotto `c` deve essere eliminato dalla lista nel caso in cui sia stata venduta l'ultima unità.

```
void vendita (Magazzino *m, char prodotto)
{
  if (*m != NULL)
    {
      Magazzino prec=NULL, corr=*m;
      boolean trovato = false;
      while(corr != NULL && !trovato)
        {
          if (corr->codice==prodotto)
            trovato = true;
          else
             {
                prec=corr;
                corr=corr->next;
             }
          if (trovato)
             {
               corr->quantita = corr->quantita - 1;
               if (corr->quantita==0)
                 {
                   if (prec==NULL)
                     *m = *m->next;
                   else
                     prec->next = corr->next;
                   free(corr);
                 }
             }
        }
      }
}
```