PROGRAMMAZIONE 1 e LABORATORIO (A,B) - a.a. 2011-2012
Prova scritta del 4/06/2012

SOLUZIONI PROPOSTE

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

ESERCIZIO 1 (punti 6)
Un albero binario si dice *quasi perfettamente bilanciato* se e solo se:

- è vuoto, oppure
- non è vuoto, il sottoalbero destro e il sottoalbero sinistro hanno profondità che differiscono al più di 1 ed entrambi sono quasi perfettamente bilanciati.

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 controlla se un albero binario è quasi perfettamente bilanciato.

Soluzione
Si propongono due soluzioni.

Prima soluzione

```
let rec qbt bt =
  let rec depth bt = match bt with
      Void -> 0 |
      Node(_,lt,rt) -> let pl = depth lt in
                          let pr = depth rt in
                             if pl>pr then 1 + pl else 1 + pr
  in
     let absd n m = if n>m then n-m else m-n
       in
         match bt with
            Void -> true |
            Node(_, lt, rt) -> (qbt lt) & (qbt rt) & absd (depth lt) (depth rt) <=1;;
```

Seconda soluzione

```
let qbt bt =
   let rec qbt_q_p bt = match bt with
          Void -> (true, 0) |
          Node(_,lt,rt) ->
              let absd n m = if n>m then n-m else m-n
              in
                let (q1, p1) = qbt_q_p lt
                and (q2, p2) = qbt_q_p rt
                in
                  (q1 & q2 & absd p1 p2 <= 1,
                   if p1 > p2 then 1+p1 else 1 + p2)
   in
     let (q,_) = qbt_q_p bt in q;;
```

ESERCIZIO 2 (punti 6)
Si definisca in `C` una funzione

```
boolean check (int a[], int b[], int dim)
```

che restituisce il valore di verità della seguente formula:

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

Suggerimento: si cerchi di comprendere il significato della proprietà espressa dalla formula.

Soluzione
La proprietà è vera se e solo se l'array `b` è lo shift circolare a destra dell'array `a` di `j` posizioni, per un valore di `j` compreso nell'intervallo `[0, dim)`.
Si propongono due soluzioni.

Prima soluzione

```
boolean check (int a[], int b[], int dim)
{
int j=0; boolean trovato = false;

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

Seconda soluzione

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

ESERCIZIO 3 (punti 6)
Si consideri la seguente grammatica $G = \langle \{1,2,3\}, \{\texttt{S}, \texttt{A}, \texttt{B}\}, \texttt{S}, P \rangle$, dove le produzioni in P sono le seguenti:

```
S ::= S3 | A
A ::= 1A | B
B ::= 2B | 2
```

Scrivere in `C` una funzione che, dato un array di interi `a` e la sua dimensione `dim` restituisce `true` se la sequenza `a[0]a[1]...a[dim-1]` appartiene al linguaggio generato da G, e restituisce `false` altrimenti.

Soluzione

Odderviamo che il linguaggio generato dalla grammatica è il seguente¿

$$\mathcal{L} = \{1^n 2^m 3^k \mid n \geq 0, k \geq 0, m > 0\}$$

```
boolean linguaggio (int a [], int dim)
{
   boolean appartiene = a[0]==1 || a[0]==2;
   int i=1;
   while (i<dim && appartiene)
     if (a[i]<=3 && ((a[i]==a[i-1]) || a[i]==a[i-1]+1)) i=i+1; else appartiene=false;
   return (appartiene && a[dim-1]>=2);
}
```

ESERCIZIO 4 (punti 6)
Si completi la seguente definizione

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

in modo che

```
tris  : int list -> int * int list * int list
```

e `(tris lis)` restituisca la tripla `(s, l1, l2)`, in cui `s` è la somma degli elementi di `lis`, `l1` è la lista di tutti gli elementi di `lis` che non sono maggiori della somma degli elementi che li seguono in `lis`, e `l2` è la lista di tutti gli elementi di `lis` che sono maggiori della somma degli elementi che li seguono in `lis`.

Soluzione

```
let tris l = let f x (s, l1, l2) =
                 if x<=s then (x+s, x::l1, l2) else (x+s, l1, x::l2)
           in  foldr f (0, [], []) 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, con la proprietà che tutti gli elementi negativi precedono quelli non negativi, inserisce x nella lista in modo da preservare la stessa proprietà.

Soluzione
Si propongono due soluzioni.

Prima soluzione

```
void inserisci (ListaDiElementi *lista, int x)
{
ListaDiElementi nuovo = malloc(sizeof(ElementoDiLista));
nuovo -> info = x;
if (x<0 || (*lista)==NULL)
    {
      nuovo -> next = *lista;
      *lista = nuovo;
    }
else
    {
      ListaDiElementi corr=*lista,
      while (corr -> next != NULL)
        corr = corr->next;}
      corr->next = nuovo;
      nuovo->next = NULL;
    }
}
```

Seconda soluzione

```
void inserisci (ListaDiElementi *lista, int x)
{
ListaDiElementi nuovo = malloc(sizeof(ElementoDiLista));
nuovo -> info = x;
if (x<0 || (*lista)==NULL)
    {
      nuovo -> next = *lista;
      *lista = nuovo;
    }
else
    {
      ListaDiElementi prec=*lista, corr=(*lista)->next;
      boolean primo_non_neg=false;
      while (corr != NULL && !primo_non_neg)
        if (corr -> info >= 0) primo_non_neg = true;
          else {prec=corr; corr= prec->next;}
      nuovo->next = corr;
      prec->next = nuovo;
    }
}
```