PROGRAMMAZIONE 1 e LABORATORIO (A,B) - a.a. 2011-2012
Prova scritta del 10/01/2012

SOLUZIONI PROPOSTE

ESERCIZIO 1 (punti 5)
Un albero *discendente a destra* (*dad*) è:

- una foglia
- un nodo con n figli ordinati (da sinistra a destra) tale che i primi $n-1$ sono foglie e l'n-esimo è la radice di un albero *dad*.

Si dia una grammatica G per il linguaggio $\{a^n b^m c^k \mid n,m,k \geq 0, n+m+k > 0\}$ sull'alfabeto $\Lambda = \{a,b,c\}$, in modo che tutti gli alberi di derivazione in G siano alberi *dad*.

Soluzione
Il linguaggio dato non deve generare la stringa vuota. Affinch gli alberi di derivazione siano alberi *dad* ogni produzione deve essere del tipo $A \to \alpha$, con $\alpha \in \Lambda^+$ oppure $A \to \alpha B$, con $\alpha \in \Lambda^*$ e B categoria sintattica. Una possibile soluzione è la seguente:

```
S ::= a | aS | B
B ::= b | bB | C
C ::= c | cC
```

ESERCIZIO 2 (punti 5)
Il taglio a profondità n di un albero binario è la lista delle etichette di tutti i nodi a profondità n, presi da sinistra a destra. Dato il tipo degli alberi binari visto a lezione

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

si definisca una funzione `cut` con tipo

```
cut : 'a btree -> int -> 'a list
```

in modo che `(cut bt n)` sia il taglio a profondità `n` di `bt`.

Soluzione

```
let rec cut bt n = match (bt, n) with
  t,m  when t=Void or n<1 -> [] |
  Node(x,_,_), 1 -> [x] |
  Node(_,lt,rt), m when m>1 -> (cut lt (n-1)) @ (cut rt (n-1));;
```

ESERCIZIO 3 (punti 5)
Si definisca in `C` una funzione

`boolean check (int a[], int d)`

che restituisce il valore di verità della seguente formula:

$$\exists i.\ i \in [1, d-1)\ \wedge \sum_{j=0}^{i-1} a[j] = \sum_{j=i+1}^{d-1} a[j]$$

(Si assuma predefinito il tipo `typedef enum {false, true} boolean`).

Soluzione
Forniamo due possibili soluzioni.

Prima soluzione

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

Seconda soluzione

```
boolean check (int a [], int d)
{
   boolean trovato = false;
   if (d>2)
   {
     int i = d-2, sommaprec = 0, sommapost=a[d-1], j;
     for(j=0; j<i; j++)
       sommaprec = sommaprec+a[j];;
     trovato = (sommaprec == sommapost);

     while(i > 1 && !trovato)
      {
       sommapost = sommapost + a[i];
       i = i-1;
       sommaprec = sommaprec - a[i];
       trovato = (sommaprec == sommapost);
      }
   }
   return trovato;
}
```

ESERCIZIO 4 (punti 5)
Si definisca in `C` una funzione

```
int max_occ (int  a[], int dim)
```

che restituisce uno dei valori che occorrono in `a` il maggior numero di volte.

Soluzione

```
boolean max_occ (int a[], int dim)
{
   int i, j, occ, val, occ_val=0;
   i=0;
   while(i<dim-occ_val)
     {
       occ = 1;
       for(j=i+1; j<dim; j++)
         if (a[i]==a[j]) occ++;
       if (occ > occ_val) {val = a[i]; occ_val = occ;}
   i=i+1;
     }
   return val;

}
```

ESERCIZIO 5 (punti 5)
Si completi la seguente definizione

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

in modo che `(revpari lis)` sia la lista degli elementi pari di `lis` in ordine inverso. Ad esempio:

```
revpari [1;5;2;4;7;8;10] = [10;8;4;2]          revpari [5;1;3] = []
```

Soluzione

```
let revpari l = let f x y = if (x mod 2=0) y @ [x]  else y in foldr f [] l;;
```

ESERCIZIO 6 (punti 5)
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 una lista di interi ed un intero x, elimina dalla lista l'ultimo elemento uguale a x e quello che lo segue immediatamente (se esistono).

Soluzione
Forniamo due possibili soluzioni.

Prima soluzione

```
void p (ListaDiElementi *l, int x)
{
ListaDiElementi scorr, prec, aux, px = NULL;
if (*l != NULL)
 if ((*l)->next != NULL)
    {
      scorr = *l;
      while (scorr != NULL)
        { if (scorr->info == x)
            px=scorr;
            scorr = scorr -> next;
        }
      if (px !=NULL)
      {
        if (px == *l)
          {  aux = (*l)->next;
             *l = (*l)->next->next;
             free(aux);
             free(px);
          }
        else
         {
             if (px -> next != NULL)
               {
                 prec = *l;
                 scorr = (*l) -> next;
                 while (scorr != px)
                 {
                   prec = scorr;
                   scorr = scorr -> next;
                 }
                 prec->next = px->next->next;
                 free(px->next);
                 free(px);
               }
         }
      }
    }
}
```

Seconda soluzione

```
void p (ListaDiElementi *l, int x)
{
ListaDiElementi scorr, prec, aux, ppx=NULL, px = NULL;
if (*l != NULL)
 if ((*l)->next != NULL)
    {
      scorr = *l;
      prec = NULL;
      while (scorr != NULL)
        { if (scorr->info == x)
               {ppx = prec; px=scorr;}
          prec=scorr;
          scorr = scorr -> next;
        }
      if (px !=NULL)
      {
        if (px == *l)
          {
            aux = (*l)->next;
            *l = (*l)->next->next;
            free(aux);
            free(px);
          }
        else
          if (px->next != NULL)
            {
              ppx->next=px->next->next;
              free(px->next);
              free(px);
            }
      }
    }
}
```