PROGRAMMAZIONE 1 e LABORATORIO (A,B)
a.a. 2009-2010
Prova scritta del 14 gennaio 2010

SOLUZIONI PROPOSTE

ESERCIZIO 1 (punti 5)
Data una grammatica $G = \langle \Lambda, V, S, P \rangle$ che genera il linguaggio $\mathcal{L}_G$ e supponendo $\downarrow \notin \Lambda$, si definisca una nuova grammatica $G' = \langle \Lambda \cup \{\downarrow\}, V', S', P' \rangle$ che genera il seguente linguaggio

$$\mathcal{L}_{G'} = \{\alpha_1 \downarrow \alpha_2 \downarrow \ldots \downarrow \alpha_n \mid n \text{ pari maggiore di 0 e } \alpha_i \in \mathcal{L}_G \text{ con } i \in [1, n]\}$$

(**N.B.** Indicare con precisione tutte le componenti di G')

Soluzione
Aggiungiamo una nuova categoria sintattica S', che è anche il simbolo iniziale di G', all'insieme V con le due nuove produzioni:

$$S' \to S \downarrow S \mid S \downarrow S \downarrow S'$$

(scegliendo opportunamente $S' \notin V$). Dunque:

$$G' = \langle \Lambda \cup \{\downarrow\}, V \cup \{S'\}, S', P \cup \{S' \to S \downarrow S, S' \to S \downarrow S \downarrow S'\} \rangle$$

ESERCIZIO 2 (punti 5)
Dato il tipo degli alberi binari visto a lezione

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

si definisca una funzione `elim` con tipo

```
elim : 'a btree -> 'a -> 'a btree
```

in modo che `(elim bt x)` sia l'albero ottenuto eliminando da `bt` tutte le foglie etichettate con `x`.

Soluzione

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

ESERCIZIO 3 (punti 5)
Si definisca in C una procedura

```
void occurr (int  a[], int dima, int b[], int dimb)
```

che sostituisce ad ogni elemento del vettore `a` (di dimensione `dima`) il numero delle sue occorrenze nel vettore `b` (di dimensione `dimb`).

Soluzione

```
void occurr (int  a[], int dima, int b[], int dimb)
{
   int i,j,conta;
   for (i=0; i<dima; i=i+1)
     {
        conta = 0;
        for(j=0; j<dimb; j=j+1)
           if (a[i]==b[j])
              conta = conta + 1;
        a[i] = conta;
     }
}
```

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

```
boolean check (int  vet[], int dim)
```

che restituisce `true` se e solo se il vettore `vet` (di dimensione `dim`) soddisfa la seguente proprietà:

$$\exists i \in [0, \texttt{dim}).\ \big(\forall j \in [0, i).\texttt{vet}[j] = \texttt{vet}[j+1]\big) \wedge \big(\forall j \in (i, \texttt{dim}-1).\texttt{vet}[j] = \texttt{vet}[j+1]\big)$$

e restituisce `false` altrimenti (si supponga dato il tipo `typedef enum {false, true} boolean;`).

Soluzione
La soluzione proposta utilizza due volte lo schema della ricerca lineare incerta per:
(1) individuare il minimo indice `i`, se esiste, per cui `vet[i]` è diverso da `vet[i+1]`;
(2) verificare che l'array sia costante nella porzione `[i+1, dim)`.
Si noti che, se in (1) non viene individuato alcun indice, l'array è costante e dunque soddisfa la proprietà richiesta.

```
boolean check (int  vet[], int dim)
{
   int i; boolean trovato;

   i=0; trovato = false;
   while (i<(dim-1) && !trovato)
      if (vet[i]<>vet[i+1])
         trovato = true;
      else
         i=i+1;

   i=i+1;
   trovato = false;

   while (i<(dim-1) && !trovato)
      if (vet[i]<>vet[i+1])
         trovato = true;
      else
         i=i+1;

   return (!trovato)
}
```

ESERCIZIO 5 (punti 5)
Si completi la seguente definizione CAML

```
let ins x l =
    let f z y = ...
in
    foldr f [x] l;;
```

in modo che, nell'ipotesi che `lis` sia una lista di elementi ordinata in modo non decrescente, il risultato di `(ins x lis)` sia la lista ottenuta inserendo `x` in `lis` e mantenendo l'ordinamento.

Soluzione
Ricordando la definizione di `foldr`, si osserva che, data una lista `l = [x1;x2; ... ; xn]`, il risultato di `foldr f [x] l` è

```
f x1 (f x2 (......  (f xn [x])...))
```

Dunque il ruolo di `f` deve essere quello di far *fluttuare* `x` nella sua corretta posizione, rispettando l'ordinamento.

```
let ins x l =
    let f z y =
         match y with
           w::ws when z<w  -> z::w::ws |
           w::ws when z>=w -> w::z::ws
in
    foldr f [x] l;;
```

ESERCIZIO 6 (punti 5)
Scrivere in C una procedura che cancella da una lista di interi il primo elemento che contiene un valore n ed è seguito da un elemento che contiene il valore $2n$, e che lascia invariata la lista se un tale elemento non esiste.
Si suppongano date le seguenti definizioni:

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

typedef struct el ElementoLista;
typedef ElementoLista *ListaDiElementi;
```

Soluzione
Osserviamo subito che la lista deve essere passata per indirizzo in quanto l'elemento da eliminare potrebbe proprio essere il primo.

```
typedef enum {false, true} boolean;

void cancellaDoppio (ListaDiElementi *lista)
   {
       ListaDiElementi corr, prec;
       boolean finito;

       if (*lista != NULL)
          if (*lista -> next != NULL)
             if (*lista -> next -> info == 2 * (*lista ->info))
                 {
                   corr = *lista;
                   *lista = *lista -> next;
                   free (corr);
                 }
              else
                 {
                    prec = *lista;
                    corr = *lista -> next;
                    finito = false;
                    while (corr -> next != NULL && !finito)
                       {
                          if (corr -> next -> info == 2 * (corr ->info))
                               {
                                  prec -> next = corr -> next;
                                  free(corr);
                                  finito = true;
                               }
                          else
                               {
                                  prec = corr;
                                  corr = corr -> next;
                               }
                       }
                 }
    }
```