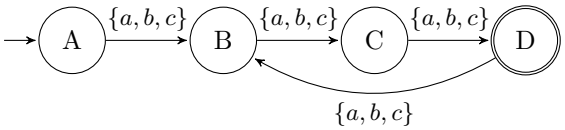
PROGRAMMAZIONE 1 e LABORATORIO (A,B) - a.a. 2015/16
Prova scritta del 4/02/2016

SOLUZIONI PROPOSTE

Per ogni esercizio vengono proposte una o più soluzioni tra le tante possibili.

ESERCIZIO 1 (6 punti)
Dato l'alfabeto $\Lambda = \{a, b, c\}$ si definisca una grammatica **regolare** che genera il linguaggio delle stringhe di Λ^+ la cui lunghezza è un multiplo di 3.

Soluzione
Costruiamo dapprima un automa a stati finiti che riconosce il linguaggio.



da cui si ricava semplicemente la seguente grammatica regolare

```
A ::= aB | bB | cB
B ::= aC | bC | cC
C ::= aD | bD | cD | a | b | c
D ::= aB | bB | cB
```

ESERCIZIO 2 (6 punti)
Si definisca in CAML, senza utilizzare ricorsione esplicita, una funzione

```
foo : 'a list -> 'a * int * int
```

in modo che `(foo lis)` restituisca una terna `(max, n, m)` in cui `max` è il valore massimo presente nella lista, `n` è il numero di occorrenze di `max` nella lista e `m` è il numero di elementi della lista. La funzione non è definita se la lista è vuota.
N.B. Non è consentito l'utilizzo della funzione `length`.

Soluzione

```
let foo l =
   let f x (max, n, m) = if x>max then (x, 1, m+1)
                                  else if x=max then (max, n+1, m+1)
                                                else (max, n, m+1)
   in
     match l with
       x::xs -> foldr f (x,1,1) xs;;
```

ESERCIZIO 3 (6 punti)
Si scriva in `C` una procedura che, dati attraverso opportuni parametri una lista di interi ed un intero `n`, sposta in testa alla lista tutti gli elementi maggiori di `n`.
Si suppongano predefiniti i tipi

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

typedef struct el ElementoDiLista;
typedef ElementoDiLista *ListaDiInteri;
```

Soluzione

```
void move (ListaDiInteri *l, int n)
{
     if (*l != NULL)
         if (*l -> next != NULL)
           {
              ListaDiInteri prec=*l, corr=*l -> next;
              while (corr != NULL)
                  if (corr -> info > n)
                    {
                      prec -> next = corr -> next;
                      corr -> next = *l;
                      *l = corr;
                      corr = prec -> next;
                    }
                  else
                    {
                      prec = corr;
                      corr = corr -> next;
                    }
           }
}
```

ESERCIZIO 4 (6 punti)
Si scriva una funzione `C`

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

che, dati due array di interi a, b e le loro dimensioni $dima, dimb$, restituisce il valore di verità della seguente formula

$$\exists i \in [0, dima).\big((\forall j \in [0, dimb).\ a[i] \neq b[j])\ \wedge\ \#\big\{k\ \mid\ k \in [0, dima) \wedge\ a[i] = a[k]\big\} = 1\big)$$

Si ricorda che, dato un insieme finito A, $\#A$ denota il numero di elementi di A.

Soluzione
Scriviamo dapprima una funzione ausiliaria che verifica la presenza di un valore in un array

```
int member (int a[], int dima, int val)
{
  int trovato=0;
  int i = 0;
  while (i < dima && !trovato)
    {
      if (a[i]==val)
         trovato = 1;
      else
         i= i+1;
    }
  return (trovato);
}


int check (int a[], int dima, int b[], int dimb)
{
   int i=0; int trovato = 0;
   while (i < dima && !trovato)
     {
       if (!member(b, dimb, a[i]))
         {
           int contaocc = 0; int j=0;
           while (j<dima && contaocc < 2)
              { if (a[j]==a[i])
                    contaocc = contaocc+1;
                j = j+1;
              }
           trovato = (contaocc == 1);
         }
       i = i+1;
     }
   return (trovato);
}
```

ESERCIZIO 5 (6 punti)
Dato il tipo degli alberi binari

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

definire una funzione

```
leafsdepth: 'a btree -> int -> int
```

in modo che `(leafsdepth bt n)` restituisca il numero di foglie dell'albero che occorrono a profondità `n`.

Soluzione

```
let rec leafsdepth bt n = match bt,n with
     Void, _ -> 0 |
     Node(_, Void, Void), 1 -> 1  |
     Node(_, lbt, rbt) -> leafsdepth lbt (n-1) + leafsdepth rbt (n-1);;
```