PROGRAMMAZIONE 1 e LABORATORIO (A,B) - a.a. 2011-2012
Esercitazione del 15/12/2011
SOLUZIONI PROPOSTE

ESERCIZIO 1 (6 punti)
Si consideri il seguente tipo di alberi binari.

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

Si definisca in `CAML` una funzione `count_one` con tipo

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

in modo che `count_one alb` restituisca il numero di nodi di `alb` che hanno esattamente un figlio.

Soluzione

```
let rec count_alb bt = match bt with
     Void -> 0 |
     Node(_, Void, Void)  -> 0 |
     Node(_, Void, rt) when rt <> Void -> 1 + count_alb rt |
     Node(_, lt, Void) when lt <> Void -> 1 + count_alb lt |
     Node(_, lt, rt) when lt<>Void && rt<>Void -> count_alb lt + count_alb rt ;;
```

ESERCIZIO 2 (6 punti)
Scrivere in `C` una funzione

```
boolean subarray (int a [], int b [], int dima, int dimb)
```

che, dati due array `a` e `b` di dimensione `dima` e `dimb` rispettivamente, restituisce `true` se l'array `a` è una sottosequenza dell'array `b`, `false` altrimenti (si assuma data la definizione `typedef enum {false,true} boolean`).
Ad esempio, dati i seguenti array `vet1` e `vet2`.

3	1	5

5	3	1	5	7	9

la chiamata `subarray(vet1, vet2, 3, 6)` deve restituire `true`. Dati invece i seguenti array `vet1` e `vet2`

3	1	5

5	3	1	7	5	9

la chiamata `subarray(vet1, vet2, 3, 6)` deve restituire `false`.

Soluzione

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

ESERCIZIO 3 (6 punti)
Scrivere in `C` una procedura

```
void foo (int a[], int dim, int x)
```

che rimpiazza con il valore `x` tutti gli elementi di `a` che precedono l'ultimo valore dispari in `a`.

Soluzione

```
void foo(int a[], int dim, int x)
   {
      int i=dim-1; int j;
      boolean trovato_dispari = false;
      while (i>0 && !trovato_dispari)
         if (a[i] % 2 == 1)
            trovato_dispari = true;
         else
            i = i-1;
      for (j=0, j<i; j=j+1)
         a[j] = x;
   }
```

ESERCIZIO 4 (6 punti)
Date le seguenti definizioni:

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

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

scrivere in C una procedura che, data in ingresso attraverso un opportuno parametro una lista di interi, elimina dalla lista tutti gli elementi che precedno l'ultimo elemento dispari della lista, liberando opportunamente la memoria corrispondente agli elementi eliminati.

Soluzione

```
void foo (ListaDiElementi *lista)
   {
      ListaDiElementi UltimoDispari = NULL, scorr = *lista;

      while (scorr != NULL)
         {
           if ((scorr -> info) % 2 == 1)
              UltimoDispari = scorr;
           scorr = scorr -> next;
         }

      if (UltimoDispari != NULL)
           while (*lista != UltimoDispari)
              {
                 scorr = *lista;
                 *lista = *lista -> next;
                 free(scorr);
              }
   }
```

ESERCIZIO 5 (6 punti)
Si supponga di estendere la sintassi dei comandi con la seguente produzione:

```
Com ::= do Com incr Ide;
```

la cui semantica (rispetto al modello senza memoria dinamica) è data dalla seguente regola:

$\mathcal{S}em_c$ `do C incr id` $\rho\ \mu = \mu'[{}^{v+1}/_{\rho(\mathtt{id})}]^{mod}$

dove $\mu' = \mathcal{S}em_c$ `C` $\rho\ \mu$

e $v = \mu'(\rho(id))$

Estendere opportunamente il tipo `Com` e la definizione della funzione `semc` nella implementazione CAML della semantica dei comandi, al fine di contemplare anche il nuovo comando.

Soluzione
Estensione del tipo `Com`

```
type com = ... |
           Do_incr of com*ide;;
```

Estensione di `semc`

```
let rec semc c (a:amb list) (m:mem list) = match c with
     ...
     Do_incr(c,x) -> let m' = semc c a m in
                        let (Def l) = search a x in
                             let (Def (ValN v)) = search m' l in
                                update_stack m' l (Def (ValN (v+1)))
```