PROGRAMMAZIONE 1 e LABORATORIO (A,B) - a.a. 2013-2014
Verifica scritta del 20/12/2013

SOLUZIONI PROPOSTE

ESERCIZIO 1 (6 punti)
Dato il tipo degli alberi binari

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

si definisca in `CAML` una funzione `maxunico` con tipo

```
maxunico : 'a btree -> 'a * bool
```

in modo che `(maxunico bt)` restituisca la coppia `(m, b)` dove `m` è l'elemento massimo di `bt` e `b` è `true` se `m` occorre in `bt` esattamente una volta, `false` altrimenti. La funzione non è definita su alberi vuoti.

Soluzione
Prima soluzione

```
let maxunico bt =
     let rec max bt = match bt with
          Node(x, Void, Void) -> x |
          Node(x, Void, rbt) when rbt <> Void -> let m = max rbt
                                                 in if x>m then x else m |
          Node(x, lbt, Void) when lbt <> Void -> let m = max lbt
                                                 in if x>m then x else m |
          Node(x, lbt, rbt) when lbt <> Void & rbt <> Void ->
                       let m1 = max lbt in
                       let m2 = max rbt in
                       let m = if m1>m2 then m1 else m2
                       in
                         if x>m then x else m
     in

     let rec occ bt x = match bt with
         Void -> 0 |
         Node(y, lbt, rbt) -> if x=y then 1+(occ lbt x)+(occ rbt x) else (occ lbt x)+(occ rbt x)

     in
       let m = max bt in (m, (occ bt m)=1);;
```

Seconda soluzione

```
let rec maxunico bt = match bt with
  Node(x, Void, Void) -> (x, true) |
  Node(x, Void, rbt) when rbt <> Void -> let (m,b) = maxunico rbt in
                                         if x>m then (x,true) else
                                          if x=m then (x, false) else (m,b)  |
  Node(x, lbt, Void) when lbt <> Void -> let (m,b) = maxunico lbt in
                                         if x>m then (x,true) else
                                          if x=m then (x, false) else (m,b)  |
  Node(x, lbt, rbt) when lbt <> Void & rbt <> Void ->
                       let (m1,b1) = maxunico lbt in
                       let (m2,b2) = maxunico rbt in
                       let (m,b) = if m1>m2 then (m1,b1)
                                   else
                                     if m2>m1 then (m2,b2)
                                     else (m1,false)
                       in
                         if x>m then (x,true) else
                         if x=m then (x, false) else (m,b);;
```

ESERCIZIO 2 (6 punti)
Senza utilizzare ricorsione esplicita, definire in CAML una funzione

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

in modo che `(foo lis)` restituisca:

- la coppia costituita dagli ultimi due elementi dispari di `lis`, se questa contiene almeno due elementi dispari
- la coppia `(0, d)` se `d` è l'unico elemento dispari di `lis`
- la coppia `(0, 0)` se `lis` non contiene elementi dispari.

Soluzione

```
let foo lis = let f x (d1,d2) = if ((d1<>0 & d2<>0) or (x mod 2 = 0)) then (d1, d2) else
                                if d2=0 then (0,x) else (x,d2)
              in   foldr f (0,0) lis;;
```

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

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

che, dati due array `a` e `b` di dimensione `dima` e `dimb` rispettivamente, calcola il valore di verità della seguente formula

$$(\forall i \in [0, dima).\exists j \in [0, dimb).a[i] = b[j]) \ \wedge \ (\forall i \in [0, dimb).\exists j \in [0, dima).a[j] = b[i])$$

Soluzione

```
int check(int a [], int b [], int dima, int dimb)
{
int ia, ib, perogni, esiste;

ia=0;
perogni = 1;

while (ia<dima && perogni)
  {
    ib=0; esiste=0;
    while (ib<dimb && !esiste)
       if (a[ia] == b[ib])
          esiste = 1;
       else ib++;
    perogni = esiste;
    ia++;
  }

ib=0;
while (ib<dimb && perogni)
  {
    ia=0; esiste=0;
    while (ia<dima && !esiste)
       if (b[ib] == a[ia])
          esiste = 1;
       else ia++;
    perogni = esiste;
    ib++;
  }

return perogni;
}
```

ESERCIZIO 4 (6 punti)
Scrivere in `C` una procedura

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

che, dati due array `a` e `b` di dimensione `dima` e `dimb` rispettivamente, azzera tutti gli elementi di `a` che occorrono in `b` più di una volta e lascia inalterati tutti gli altri elementi di `a`.

Soluzione
Prima soluzione

```
void azzera(int a [], int b [], int dima, int dimb)
{
int ia, ib, conta;

for(ia=0; ia<dima; ia++)
  {
    conta=0;
    for (ib=0; ib<dimb; ib++)
      if (a[ia]==b[ib]) conta++;
    if (conta > 1) a[ia]=0;
  }
}
```

Seconda soluzione

```
void azzera(int a [], int b [], int dima, int dimb)
{
int ia, ib, conta;

for(ia=0; ia<dima; ia++)
    if (a[ia] != 0)
      {
        ib=0; conta=0;
        while (ib<dimb && conta<2)
          { if (a[ia] == b[ib]) conta++;
            ib++;
          }
        if (conta==2)
          {
            int ka;
            for (ka=ia+1; ka<dima; ka++)
              if (a[ka]==a[ia]) a[ka]=0;
            a[ia]=0;
          }
      }
}
```

ESERCIZIO 5 (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 il penultimo elemento se è uguale all'ultimo, lascia la lista invariata in ogni altro caso.

Soluzione

```
void canc_penultimo (ListaDiInteri *l)
   {
      if (*l != NULL)
        if (*l->next != NULL)
          {
            ListaDiInteri ultimo, penultimo, terzultimo;
            terzultimo = NULL;
            penultimo= *l;
            ultimo = *l -> next;

            while (ultimo -> next != NULL)
               { terzultimo = penultimo;
                 penultimo = ultimo;
                 ultimo = ultimo -> next; }

            if (penultimo -> info == ultimo -> info)
               { if (terzultimo == NULL)
                  *l = ultimo;
                 else
                  terzultimo -> next = ultimo;

                 free(penultimo); }
          }
   }
```