PROGRAMMAZIONE 1 e LABORATORIO (A,B) - a.a. 2010-2011
II Verifica scritta - 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 `replace` con tipo

```
replace : 'a btree -> 'a ->  'a btree
```

in modo che `(replace bt y)` sia l'albero ottenuto da `bt` rimpiazzando con `y` le etichette dei nodi non foglia che hanno un solo sottoalbero non vuoto.

Soluzione

```
let rec replace bt y = match bt with
    Void -> Void
 |  Node(_, Void, Void) -> bt
 |  Node(_, Void, rt) when rt<>Void -> Node(y, Void, replace rt y)
 |  Node(_, lt, Void) when lt<>Void -> Node(y, replace lt y, Void)
 |  Node(x, lt, rt) when lt<>Void and rt<>Void -> Node(x, replace lt y, replace rt y);;
```

ESERCIZIO 2 (6 punti)
Scrivere in C una funzione

```
int sumOdd (int a [], int from, int to, int dim)
```

che calcola la somma degli elementi dispari dell'array `a` (di dimensione `dim`) con indice i tale che `from` $\leq i \leq$ `to`
Ad esempio, dato l'array `vet` seguente

20	4	25	11	5	2	9

la chiamata `sumOdd(vet,1,4,7)` deve restituire il valore `41`, la chiamata `sumOdd(vet,4,0,7)` il valore `0`.

Soluzione

```
int sumOdd (int a [], int from, int to, int dim)
{
     int i, somma = 0;
     if (from < 0) from = 0;
     if (to > dim-1) to = dim-1;
     for (i=from; i<= to; i=i+1)
        if (a[i]%2==1) somma = somma + a[i];
     return somma;
 }
```

ESERCIZIO 3 (6 punti)
Scrivere in C una procedura

```
void duplicateMax (int a[], int dim)
```

che rimpiazza tutti gli elementi successivi all'ultima occorrenza del valore massimo con il valore massimo stesso.

Soluzione

```
void duplicateMax (int a[], int dim)
 {
     int indmax=0, i;
     for (i=1; i<dim; i=i+1)
         if (a[i]>=a[indmax]) indmax = i;
     for(i=indmax+1; i<dim; i=i+1)
         a[i] = a[indmax];
 }
```

ESERCIZIO 4 (6 punti)
Date le seguenti definizioni:

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

typedef struct el ElementoLista;
typedef ElementoLista *ListaDiInteri;
```

scrivere in C una procedura

```
void maxmin (ListaDiInteri lista, int *pmax, int *pmin)
```

che lascia nelle variabili puntate da `pmax` e `pmin` rispettivamente il valore massimo e il valore minimo presenti nella lista passata come primo argomento (se la lista è vuota le variabili puntate dal secondo e dal terzo argomento devono rimanere inalterate).

Soluzione

```
void maxmin (ListaDiInteri lista, int *pmax, int *pmin)
 {
    if (lista != NULL)
  {
    int max = lista -> info;
    int min = lista -> info;
    lista = lista -> next;
    while (lista != NULL)
     {
      if ((lista -> info) > max)
          max = lista -> info;
      if ((lista -> info) < min)
          min = lista -> info;
     }
    *pmax = max;
    *pmin = min;
  }
 }
```

ESERCIZIO 5 (6 punti)
Definire, senza utilizzare ricorsione esplicita e dunque utilizzando funzioni di ordine superiore, una funzione

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

in modo che `foo l` sia la lista ottenuta da `l` azzerando tutti gli elementi che sono preceduti da un numero negativo. Ad esempio

```
foo [1 ; -2; -4; 5; 6; -7; 8] = [1; -2; 0; 0; 6; -7; 0]
```

Soluzione
Proponiamo due soluzioni.

```
let foo l =
     let f x y = match y with
       [] -> [x]
     | w::ws -> if x < 0 then x::0::ws else x::w:ws
in foldr f [] l;;
```

```
let foo l =
     let f x y = match (x,y) with
       (z,ws) when z>=0 || w=[] -> z :: w
     | (z,w::ws) when z<0 -> z::0::ws
in foldr f [] l;;
```