PROGRAMMAZIONE 1 e LABORATORIO (A,B) - a.a. 2012-2013
Esercitazione del 18/12/2012
Soluzioni proposte

ESERCIZIO 1
Dato il tipo degli alberi binari

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

si definisca in CAML una funzione `check` con tipo

```
check : 'a btree -> int -> bool
```

in modo che `(check bt n)` restituisca `true` se la profondità di `bt` è al più `n`, `false` altrimenti.

Soluzione

```
let check bt n =
   let rec prof bt = match bt with
      Void -> 0  |
      Node(_, lt, rt) -> let p1 = prof lt in
                         let p2 = prof rt in
                           if p1>p2 then 1+p1 else 1+p2
   in
      (prof bt) <= n;;
```

ESERCIZIO 2
Scrivere in C una funzione

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

che, dati due array `a` e `b` di dimensione `dima` e `dimb` rispettivamente, verifica che ogni elemento di `a` con indice `i` occorra in `b` con indice `j>i`.

Soluzione

Prima soluzione

```
boolean check (int a [], int b [], int dima, int dimb)
{
int i=0;
int j;
boolean trovato = false;

while (i<dima && !trovato)
  {
    boolean esiste = false;
    j=i+1;
    while (j < dimb && !esiste)
      if (a[i]==b[j]) esiste = true;
         else j=j+1;
    if (esiste) i=i+1; else trovato=true;
  }
return (!trovato);
}
```

Seconda soluzione

```
boolean check (int a [], int b [], int dima, int dimb)
{
int i=0;
int j;
boolean trovato = false;

while (i<dima && !trovato)
  {
    j=i+1;
    while (j < dimb && !trovato)
      if (a[i]==b[j]) trovato = true;
         else j=j+1;
    if (trovato) i=i+1;
    trovato = !trovato;
  }
return (!trovato);
}
```

ESERCIZIO 3
Scrivere in `C` una procedura

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

che rimpiazza ogni elemento `x` dell'array `a` con il numero di elementi di `a` uguali a `x` che lo precedono.

Soluzione

```
boolean replace (int a[], int dima)
{
int i, j, conta;

for (i=dim-1; i>=0; i=i-1)
  {
    conta=0;
    for (j=0; j<i; j=j+1;)
      if (a[i]==a[j]) conta=conta+1;
    a[i]=conta;
  }
}
```

ESERCIZIO 4
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, modifica la lista portando nella parte iniziale della stessa tutti gli elementi negativi (**N.B.**: non è consentito utilizzare `malloc`).

```
void p (ListaDiElementi *lis)
{
   if ((*lis) != NULL)
     {
        if ((*lis)->next != NULL)
          {
            ListaDiElementi prec, corr;
            prec = NULL;
            corr = *lis;
            while (corr != NULL)
              {
                 if (corr->info < 0)
                      if (prec != NULL)
                        {
                          prec->next = corr->next;
                          corr->next = *l;
                          *l = corr;
                          corr = prec -> next;
                        }
                      else
                        {
                          prec = corr;
                          corr = prec -> next
                        }
                 else
                   {
                     prec = corr;
                     corr = prec -> next
                   }
              }
          }
     }
}
```

ESERCIZIO 5
Senza utilizzare ricorsione esplicita, definire in CAML una funzione

```
foo : int list -> (int -> bool) ->  (int * int)
```

in modo che `(foo lis p)` restituisca la coppia `(somma,num)` dove `somma` è la somma degli elementi di `lis` che soddisfano `p` e `num` è il numero di elementi di `lis` che non soddisfano `p`.

Soluzione

```
let foo lista p =
     let
        f x (s,n) = if (p x) then (s+x, n) else (s, n+1)
     in
        foldr f (0,0) lista
```