PROGRAMMAZIONE 1 e LABORATORIO (A,B) - a.a. 2013-2014
Esercitazione del 17/12/2013

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 il numero di nodi non foglia di `bt` è maggiore di `n`, `false` altrimenti.

ESERCIZIO 2
Senza utilizzare ricorsione esplicita, definire in CAML una funzione

```
foo : 'a list -> ('a -> bool) -> 'a -> ('a * 'a)
```

in modo che `(foo lis p x)` restituisca la coppia `(x1, x2)` dove `x1` è il primo elemento di `lis` che soddisfa `p` e `x2` è l'ultimo elemento di `lis` che soddisfa `p`, oppure la coppia `(x,x)` se nessun elemento di `lis` soddisfa `p`. Si assuma che valga `not (p x)`.
Supponendo ad esempio che sia definito il predicato `pari: int -> bool` con l'ovvio significato:

```
foo [3 ; 10; 20; 5; 40; 3] pari 1 = (10, 40)
foo [3 ; 10; 11; 5; 7; 3] pari 1 = (10, 10)
foo [3 ; 1; 11; 5; 7; 3] pari 1 = (1, 1)
```

ESERCIZIO 3
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

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

ESERCIZIO 4
Scrivere in `C` una procedura

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

che azzera ogni elemento dell'array che sia preceduto da un elemento minore.
Dato ad esempio il seguente array `vet1`

10	8	15	7	3	20	5	3

la chiamata `azzera(vet1,8)` deve modificare `vet1` come segue

10	8	0	7	3	0	0	3

ESERCIZIO 5
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, se esistono, i primi due elementi consecutivi uguali tra loro.

```
let check bt m =
    let rec contenodi bt1 = match bt1 with
        Void -> ∅
      | Node (x, Void, Void) -> ∅
      | Node (x, lbt, rbt) when lbt <> Void or rbt <> Void
                -> 1 + contenodi lbt + contenodi rbt
    in contenodi bt > m ;;
```

```
let foo l p x =
    let f w (x1, x2) =
        if p w then if p x1 then (x1 x) (x2 x) (p x2)
                    else (w, x2)
               else (w, w)
    in fold f (x, x) l ;;

val not : ('a -> bool) -> 'a -> bool
```

```
int check (int a[], int b[], int dima, int dimb)
{ int ia=0; int perogni=1;
  while (ia<dima && perogni)
  { int ib=dimb-1; int trovato=0;
    while (ib>=0 && !trovato)
      if (a[ia]==b[ib]) trovato=1;
          else ib=ib-1;
    perogni = !trovato || (trovato && ib>ia);
  }
return perogni;
}
```

```
void azzera (int a[], int dim)
{ int i;
  for (i = dim-1; i>0; i--)
    { int j = i-1; int trovato = 0;
      while (j>=0 && !trovato)
        if (a[i] > a[j]) trovato = 1;
        else j=j-1;
      if (trovato) a[i] = 0;
    }
}
```

```
void canc ( ListeDiInteri * l )
{ if (*l != NULL)
    if (*l -> next != NULL)
      { if (*l -> info == *l -> next -> info)
          { ListeDiInteri aus = *l; *l = *l -> next -> next;
            free (aus -> next); free (aus); }
        else { ListeDiInteri prec = *l; ListeDiInteri corr = *l -> next;
               int trovati = 0;
               while (corr -> next != NULL && !trovati)
                 if (corr -> info == corr -> next -> info) trovati = 1;
                 else { prec = corr; corr = corr -> next; }
```

```
        if (trovati) { prec->next = corr->next->next;
                       free(corr->next); free(corr);}
    }
  }
}
```