```
//
// PRE-APPELLO: possibili soluzioni
//

/* ESERCIZIO 1
Scrivere una funzione che calcoli la media degli elementi contenuti
 in un array di numeri e che rispetti la seguente interfaccia:
float avg(float A[], int size)
utilizzare il meccanismo della ricorsione (e' possibile realizzare funzioni
 ausiliarie)
 */

/* VERSIONE 1 - Senza ausiliarie */
/* NB: per calcolare la media un elemento alla volta, occorre
        utilizzare un'opportuna pesatura */

float avg(int A[], int size) {
    string s="some error";
    if (size==0) throw s;
    if (size==1) return A[0];
    return (A[size-1]+(size-1)*avg(A,size-1))/size;
}

/* VERSIONE 2 - Con funzione ausiliaria che calcola la somma */
int sum(int A[], int size) {
    string s="some error";
    if (size==0) throw s;
    if (size==1) return A[0];
    return A[size-1]+sum(A,size-1);
}

float avg(int A[], int size) {
    string s="some error";
    if (size==0) throw s;
    return sum(A,size)/float(size);
}


/* ESERCIZIO 2
Realizzare una funzione che cancelli
tutte le istanze di un dato valore da una lista.
*/

typedef int Elem;
typedef struct cell {
    Elem head;
    cell *next;
} *lista;

/* VERSIONE 1 - Iterativa */

void delete_all_elem(Elem x, lista &l)
{
    lista cur=l;
    lista prev;
```

```
    if (l)
        return;
    else {
        while (cur!=nullptr) { //visito comunque fino in fondo
            if (cur->head==x)
            {
                if (cur==l){ //l'elemento si trova in testa
                    l=l->next;
                    delete cur;
                    cur=l;
                }
                else { // in tutti gli altri casi
                    prev->next=cur->next;
                    delete cur;
                    cur=prev->next;
                }
            }
            else {
                prev=cur;
                cur=cur->next;
            }
        }
    }
}

/* VERSIONE 2 - Ricorsiva */

void delete_all_elem(Elem x, lista &l)
{
    if (l == nullptr)  return;
    if (l->head == x)
            {
            cell *tmp = l;
            l = tmp->next;
            delete tmp;
            delete_all_elem(x,l);
            }
    else
         delete_all_elem(x, l->next);

}

/* ESERCIZIO 3 Considerare l'implementazione di Point e Rect vista a lezione

    Realizzare una funzione che restituisca l'area del minore tra due Rect
     passati come parametro. Prevedere, se ritenuto utile, funzioni
     ausiliarie
 */

struct Point { double x; double y; };
struct Rect {
    Point top_left;
    Point bottom_right;
};
```

```
/* assumo che i Rect siano ben costruiti */

float Area(Rect R) {
    float b = R.bottom_right.x - R.top_left.x;
    float h = R.top_left.y - R.bottom_right.y;
    return b*h;
}

float MinArea(Rect R1, Rect R2) {
    float a1 = Area(R1);
    float a2 = Area(R2);
    if (a1<a2) return a1; else return a2;
}

/*
  4A. La situazione  schematizzata
 descrive correttamente un possibile array_dinamico?


 RISPOSTA: NO. la porzione di memoria associata al puntatore d deve essere
  allocata nello HEAP per poterla gestire in maniera dinamica

 4B. Realizzare una funzione init_ad di inizializzazione che permetta di
  costruire un array_dinamico ancora vuoto */

struct array_dinamico {
    int *d;
    int size;
    int capacity;
};

// VERSIONE 1: creo un array dinamico vuoto - nessun elemento in memoria
 gli e' ancora associato, ma tutti i campi sono correttamente inizializzati

array_dinamico init_ad() {
    array_dinamico ad;
    ad.d = nullptr;
    ad.size=0;
    ad.capacity=0;
}

// VERSIONE 2 che ho comunque considerato corretta per ambiguità della
 domanda: creo un array dinamico con size=0 e capacity diversa da 0, con
 area allocata in memoria di dimenzione "capacity"

array_dinamico init_ad(int capacity) {
    if (capacity < 0) throw some_error;
    array_dinamico ad;
    ad.d = new int [capacity];
    ad.size=0;
    ad.capacity=capacity;
}
```