ESERCIZIO 1

1.A (1+1 PUNTI) Realizzare due struct ***indirizzo*** e ***cliente***:

- ***Indirizzo*** contiene: via, numero civico, CAP, città
- **Cliente** contiene: cognome, nome, indirizzo

1.B (1 PUNTO) Realizzare una funzione che verifichi se due clienti abitano nella stessa zona della città

1.C (1 PUNTO) ***Realizzare il prototipo*** di una funzione che acquisisca gli opportuni parametri formali, passati in modo adeguato, per aggiornare l'indirizzo di un cliente

ESERCIZIO 2 Assumiamo di voler implementare il tipo di *dato "coda di clienti"* basandoci sui **vector**

2.A (2 PUNTI) Produrre i prototipi (o interfacce) delle 3 funzioni principali
- *enqueue* (inserisci elemento in fondo alla coda)
- *dequeue* (elimina elemento dalla testa della coda)
- *front* (accedi in lettura e restituisci elemento nella coda)

2.B (2.5 PUNTI) Implementare la funzione
PARI *dequeue*
DISPARI *enqueue*
trattando opportunamente il caso coda vuota

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

ESERCIZIO 3 Considerate le liste collegate semplici (ordinate):

3.A (2.5 PUNTI) Realizzare una funzione **ricorsiva** che
PARI inserisca un elemento in ordine in una lista
DISPARI cancelli un elemento da una lista ordinata

3.B (2 PUNTI) Realizzare una funzione **booleana** che restituisce true se due liste
PARI sono consecutive (il primo elemento della 2a lista è maggiore dell'ultimo della 1a)
DISPARI sono uguali

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

}
```

```
void insert_elem(int x, lista &l)
{
    if ((l == nullptr) || (l->head > x)) //base
    {
    // inserisci in testa
    cell * aux = new cell;
    aux->head = x; aux->next = l; l = aux;
    }
    else
        insert_elem(x,l->next); // inserisci nel resto
}
```