Note
È considerato errore qualsiasi output non richiesto dagli esercizi. È consentito utilizzare funzioni ausiliarie per risolvere gli esercizi.
È importante scrivere il proprio main in Visual Studio per poter fare correttamente il debug delle funzioni realizzate!

Esercizio 1 (punti 6)
Nel file `inverti.c` definire la funzione corrispondente alla seguente dichiarazione:

```
extern void Inverti(int *v, int v_size);
```

La funzione accetta un vettore di interi (`v`), anche vuoto, e la sua dimensione (`v_size`) e deve scambiare ricorsivamente gli elementi di `v` per posizionarli in ordine inverso rispetto alla posizione iniziale. Se, ad esempio, il vettore `v` contenesse gli elementi `0 1 2 3 4 5`, al termine dell'esecuzione della funzione `Inverti` dovrebbe contenere gli elementi `5 4 3 2 1 0`. Si consiglia l'utilizzo di una funzione ausiliaria per realizzare la ricorsione. **N.B. Non sarà considerata valida nessuna soluzione che non faccia uso della ricorsione per scorrere gli elementi del vettore.**

Esercizio 2 (punti 7)
Nel file `filtra.c` definire la funzione corrispondente alla seguente dichiarazione:

```
extern list Filtra(list l, const char *citta);
```

La funzione prende in input una lista di indirizzi (anche vuota) e una citta e costruisce una nuova lista contenente tutti gli indirizzi che non fanno parte di quella città. Gli elementi della lista di output devono essere nello stesso ordine di quella di input. Se la lista è una lista vuota la funzione deve ritornare NULL. **N.B. quando fate la sottomissione della soluzione non dovete caricare il codice delle primitive, ma solo il codice della funzione Filtra e di eventuali funzioni ausiliarie.**

Per la risoluzione di questo esercizio avete a disposizione le seguenti definizioni:

```
typedef struct indirizzo {
  char via[50]; int civico; char citta[30];
} indirizzo;
typedef indirizzo element;
typedef struct list_element {
  element value;
  struct list_element *next;
} item;
typedef item* list;
```

e le seguenti primitive:

```
list EmptyList();
list Cons(const element *e, list l);
bool IsEmpty(list l);
element Head(list l);
list Tail(list l);
element Copy(const element *e);
void FreeList(list l);
list InsertBack(list l, const element *e);
```

trovate le definizioni, le dichiarazioni e le rispettive implementazioni nei file `liste_indirizzo.h` e `liste_indirizzo.c`. La documentazione delle primitive sopraelencate è disponibile al link:

http://imagelab.ing.unimore.it/olj2/Esami/materiale/20192805_liste_indirizzo/liste__indirizzo_8h.html

Esercizio 3 (punti 8)
Nel file `sort.c` definire la funzione corrispondente alla seguente dichiarazione:

```
extern bool QuickSort(int *v, int v_size, int first, int last);
```

La funzione prende in input un vettore di interi `v`, la sua dimensione `v_size` e due indici `first` e `last` e deve ordinare in ordine crescente gli elementi di `v` che si trovano tra l'indice `first` e l'indice `last` compresi. Se l'indice `first` è minore di `0`, se l'indice `last` è maggiore o uguale a `v_size`, se `last` è minore di `first`, o se `v` è `NULL` la funzione deve ritornare `false` e non modificare il vettore di input. In caso contrario la funzione deve ritornare `true` al termine dell'ordinamento. La funzione deve utilizzare l'algoritmo di ordinamento quicksort. **N.B. non sarà considerata valida nessuna soluzione che ordini il vettore con un algoritmo di ordinamento diverso dal quicksort.**

Esercizio 4 (punti 12)
Nel file `sottogruppi.c` definire la funzione corrispondente alla seguente dichiarazione:

```
extern int Sottogruppi(const char *filename, int k);
```

La funzione prende in input il nome di un file di testo (`filename`) e un intero `k`. Il file contenente i nomi degli alunni di una classe separati da `\n`. Il file contiene in totale `n` nomi (i nomi sono sequenze di caratteri che non contengono spazi) i quali sono costituiti al massimo da 19 caratteri. Il numero `n` non è noto a priori, ma si deve determinare leggendo il file. La funzione deve aprire il file e leggerne il contenuto e, utilizzando un algoritmo di backtracking ricorsivo, individuare tutti i possibili sottogruppi di `K` studenti con i seguenti vincoli:

1. due nomi che iniziano con la stessa lettera non devono mai appartenere allo stesso sottogruppo. Esempio: Andrea e Aldo hanno la stessa iniziale "A", quindi non possono appartenere allo stesso sottogruppo.
2. due nomi con iniziali consecutive non devono mai appartenere allo stesso gruppo. Esempio: Bianca e Alfredo hanno iniziali consecutive nell'alfabeto, quindi non possono mai appartenere allo stesso sottogruppo.
3. l'ordine è significativo. Esempio (`k = 2`) i gruppi { Andrea, Carlo } e { Carlo, Andrea } sono da considerarsi gruppi distinti e quindi dovrebbero comparire entrambi nella soluzione.

La funzione deve stampare su standard output tutti i possibili sottogruppi, uno per riga con il seguente formato:

```
{ Nome1, Nome2, …, NomeK }
```

La funzione deve quindi ritornare il numero di soluzioni trovate. Se non è possibile aprire il file o se il valore `k` in input è minore o uguale a `0` o maggiore di `n` la funzione deve ritornare `-1`.

Si consiglia di utilizzare una funzione ausiliaria per implementare il backtracking. Si consiglia inoltre di leggere i dati da file fuori dalla funzione ricorsiva.

Esempio:
File `es5_input1.txt`

Anna
Giovanni
Francesco

Giorgio
Bianca

Con `k = 2`, l'output dovrebbe essere:

```
{ Anna, Giovanni }
{ Anna, Francesco }
{ Anna, Giorgio }
{ Giovanni, Anna }
{ Giovanni, Bianca }
{ Francesco, Anna }
{ Francesco, Bianca }
{ Giorgio, Anna }
{ Giorgio, Bianca }
{ Bianca, Giovanni }
{ Bianca, Francesco }
{ Bianca, Giorgio }
```

N.B. l'ordine della stampa in output potrebbe variare.