Note importanti:

- È considerato errore qualsiasi output non richiesto dagli esercizi.
- È consentito utilizzare funzioni ausiliarie per risolvere gli esercizi.
- Quando caricate il codice sul sistema assicuratevi che siano presenti tutte le direttive di `include` necessarie, comprese quelle per l'utilizzo delle primitive**.** Non dovete caricare l'implementazione delle primitive.
- **È importante scrivere il proprio main in Visual Studio per poter fare correttamente il debug delle funzioni realizzate!**

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

```
extern tree Tree2Bst(tree t);
```

La funzione prende in input un albero binario di interi `t`. La funzione deve attraversare l'albero `t` utilizzando una visita in pre-ordine e costruire con i suoi elementi un nuovo albero binario BST (Binary Search Tree) sempre di interi con il seguente criterio: i figli di sinistra di un nodo devono essere sempre minori del padre, i figli di destra devono essere maggiori o uguali al padre. Se l'albero `t` è vuoto la funzione deve ritornare un BST vuoto.
Si suggerisce di leggere attentamente la documentazione delle primitive fornite prima di procedere con l'implementazione.

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

```
typedef int element;
typedef struct tree_element {
	element value;
	struct tree_element *left, *right;
} node;
typedef node* tree;
```

e le seguenti primitive:

```
tree EmptyTree();
tree ConsTree(const element *e, tree l, tree r);
bool IsEmpty(tree t);
element *GetRoot(tree t);
tree Left(tree t);
tree Right(tree t);
bool IsLeaf(tree t);
tree InsertBinOrd(const element *e, tree t);
```

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

http://imagelab.ing.unimore.it/OLJ2/esami/materiale/20191006_esame/tree_int/tree__int_8h.html

Esercizio 2 (punti 8)
Nel file `stringhe.c` definire la procedura corrispondente alla seguente dichiarazione:

```
extern void BacktrackStr(int n);
```

la procedura prende in input un intero `n` e deve stampare a video (standard output), utilizzando un approccio di backtracking, tutte le possibili stringhe di lunghezza `n` che è possibile costruire utilizzando le prime `n` lettere minuscole dell'alfabeto inglese. L'output dovrà avere il seguente formato:

```
{ xx…x }, { xx…x }, { xx…x }, … { xx…x },
```

Se `n<=0` o `n>26` la funzione non deve stampare nulla. Le stringhe devono essere stampate in ordine alfabetico.

Si consiglia di utilizzare una funzione ausiliaria per la risoluzione dell'esercizio. **N.B. Non sarà considerata valida nessuna soluzione che non faccia uso del backtracking.**

Esempio:
input: `n = 2`
output: `{ aa }, { ab }, { ba }, { bb },`

Esercizio 3 (punti 6)
Nel file `sort.c` definire la funzione corrispondente alla seguente dichiarazione:

```
extern bool BubbleSort(int *v, int v_size);
```

La funzione prende in input un vettore di interi `v` e la sua dimensione `v_size` e deve ordinare in ordine crescente gli elementi di `v`. Se `v` è `NULL` oppure se la dimensione del vettore non è strettamente maggiore di 0 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 bubble sort. **N.B. non sarà considerata valida nessuna soluzione che ordini il vettore con un algoritmo di ordinamento diverso dal bubble sort.**

Esercizio 4 (punti 5)
Sia data la seguente definizione:

```
typedef struct coords {
	int x;
	int y;
} coords;
```

Nel file `leggi.c` definire la funzione corrispondente alla seguente dichiarazione:

```
extern list LoadList(const char *filename);
```

La funzione prende in input un nome di file di testo, `filename`, contenente i dati di un poligono, ovvero un elenco di coordinate 2D rappresentati i suoi vertici. Ogni riga del file contiene 2 numeri interi separati da spazi che rappresentano rispettivamente le coordinate x e y di un vertice. La funzione deve aprire il file in modalità lettura, leggere le coordinate e aggiungerle in testa ad una lista appositamente creata. Ogni elemento della lista deve rappresentare un vertice del poligono. La funzione deve quindi restituire la lista ottenuta. Se non è possibile aprire il file o se il file è vuoto la

funzione deve ritornare una lista vuota. Si può assumere che il file sia correttamente formato, ovvero non occorre fare controlli per verificarne la correttezza.

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

```
typedef struct coords {
	int x;
	int y;
} coords;
typedef coords 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 `list_coords.h` e `list_coords.c`. La documentazione delle primitive sopraelencate è disponibile al link:

http://imagelab.ing.unimore.it/olj2/esami/materiale/20191006_esame/list_coords/list__coords_8h.html

Esercizio 5 (punti 8)
Nel file `taglia.c` definire la funzione corrispondente alla seguente dichiarazione:

```
extern list Taglia(list l, int n);
```

La funzione prende in input una lista di vertici (anche vuota) e un numero intero `n`. La funzione deve modificare la lista eliminando tutti gli elementi a partire dall'n-esimo compreso. La funzione deve ritornare la lista ottenuta dopo l'eliminazione. Se `n` è minore o uguale a 0 oppure se `n` è maggiore del numero di elementi presenti nella lista, deve essere ritornata la lista originale. Ovviamente, se `l` è una lista vuota o se `n=1` deve essere ritornata una lista vuota. La memoria degli elementi eliminati deve essere liberata! **N.B. non saranno considerate valide soluzioni che risolvono l'esercizio costruendo una nuova lista priva degli elementi da eliminare.**

Per la risoluzione di questo esercizio avete a disposizione le stesse definizioni e le stesse primitive fornite per l'esercizio 4.