Note importanti:

- È considerato errore qualsiasi output non richiesto dagli esercizi.
- È consentito, e in alcuni casi indispensabile, 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 5)
Nel file `binomiale.c` definire la funzione corrispondente alla seguente dichiarazione:

```
extern int Binomiale(int n, int k);
```

La funzione prende in input due numeri interi `n` e `k` e deve ritornare il coefficiente binomiale $\binom{n}{k}$ calcolato utilizzando la ricorsione. Se la funzione riceve in input un valore `n` minore di `0` deve ritornare `-1`. La formula ricorsiva per il calcolo del coefficiente binomiale è la seguente:

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & , \quad per\ 1 \leq k \leq n-1 \\ 1 & , \quad per\ k = 0\ o\ k = n \\ 0 & , \quad per\ k > n\ o\ k < 0 \end{cases}$$

Si consiglia di utilizzare una funzione ausiliaria per la risoluzione dell'esercizio. **N.B. Non saranno considerate valide soluzione che non fanno uso della ricorsione per il calcolo del coefficiente binomiale.**

Esercizio 2 (punti 6)
Nel file `sortlist.c` definire la funzione corrispondente alla seguente dichiarazione:

```
extern list SortList(list l);
```

La funzione prende in input una lista di interi `l` e deve ordinarne i suoi elementi per valore crescente e ritornare la testa della lista ordinata. Se la lista `l` è vuota deve essere ritornata una lista vuota. Non ci sono vincoli sull'algoritmo di ordinamento, ovvero potete scegliere l'algoritmo che preferite. Suggerimento: per scambiare due elementi della lista è sufficiente scambiarne i valori, senza dover aggiornare i puntatori.

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

```
typedef int 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_int.h` e `list_int.c`. La documentazione delle primitive sopraelencate è disponibile al link:

http://imagelab.ing.unimore.it/OLJ2/esami/materiale/20192406_esame/liste_int/list__int_8h.html

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

```
extern bool SaveTree(tree t, const char *filename);
```

La funzione prende in input un albero di interi `t` e il nome di un file `filename`. Dopo aver aperto il file in modalità scrittura non tradotta, la funzione deve scorrere l'albero in pre-ordine e scrivere sul file i valori contenuti nei suoi nodi. I numeri devono essere scritti in formato testuale decimale, nell'ordine in cui vengono visitati e separati da `<spazio>`. Se la scrittura va a buon fine la funzione deve ritornare `true`. Se non è possibile aprire il file la funzione deve ritornare `false`.

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/20192406_esame/tree_int/tree__int_8h.html

Esercizio 4 (punti 7)
Nel file `concatena.c` definire la funzione corrispondente alla seguente dichiarazione:

```
extern list Concatena(list l1, list l2, list l3);
```

La funzione prende in input tre liste `l1`, `l2`, `l3` che deve concatenare a partire dalla prima. Le liste possono essere liste vuote. La funzione deve ritornare la lista risultante, ovvero l'indirizzo della sua testa.

Se tutte le liste sono vuote, la funzione deve ritornare una lista vuota.

Per la risoluzione di questo esercizio avete a disposizione le stesse definizioni e le stesse primitive fornite per l'esercizio 2.

Esercizio 5 (punti 9)
Siano dati n sciatori di altezza $a_1, a_2, \dots, a_n$ e n paia di sci di lunghezza $s_1, s_2, \dots, s_n$. Si vuole assegnare ad ogni sciatore un paio di sci, in modo da minimizzare la differenza totale fra le altezze degli sciatori e la lunghezza degli sci. Quindi, se allo sciatore i-esimo viene assegnato il paio di sci $h(i)$, occorre minimizzare la seguente quantità:

$$\sum_{i=1}^{n} | a_i - s_{h(i)} |$$

Creare i file `sciatori.h` e `sciatori.c` che consentano di definire la seguente struttura:

```
typedef struct sciatore {
    double a;
    double l;
}sciatore;
```

e la funzione:

```
extern sciatore* Accoppia(double *altezze, double *lunghezze, int size);
```

La `struct` consente di rappresentare l'accoppiamento sciatore-sci memorizzando rispettivamente l'altezza dello sciatore `a` e la lunghezza degli sci che gli sono stati assegnati `l`. La funzione accetta come parametri due vettori di `double`, uno di altezze e uno di lunghezze, e la loro dimensione (`size`) e deve accoppiare altezze e sci utilizzando un algoritmo greedy. L'algoritmo greedy deve scegliere ad ogni passo la coppia (altezza, lunghezza) con la minima differenza (in valore assoluto). Ovviamente un'altezza deve essere accoppiata con una e una sola lunghezza e viceversa. La funzione deve salvare le coppie, nell'ordine in cui vengono scelte, in un vettore di `struct sciatore` opportunamente allocato e ritornarlo.

Il vettore di altezze e quello di lunghezze hanno sempre la stessa dimensione, indentificata dal parametro `size`. Se `size` è `0` la funzione deve ritornare NULL.