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 8)

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

```
typedef struct polinomio {
	int *coeffs;
	size_t size;
} polinomio;
```

e la funzione:

```
extern double Secanti(const polinomio *p, double x0, double x1, double t, int max_iter);
```

La `struct` consente di rappresentare una funzione polinomiale di grado n a coefficienti interi. I coefficienti del polinomio sono memorizzati a partire da quello di grado minore nel vettore `coeffs`. La dimensione del vettore `coeffs` è memorizzata nel campo `size` della `struct`. Dato ad esempio il polinomio $x^3 - 2x$, la `struct` che lo rappresenta conterrà `4` nel campo `size` e `coeffs` punterà ad un'area di memoria contenente i seguenti valori `0,-2,0,1`.

La funzione deve applicare ricorsivamente il metodo delle secanti per calcolare in modo approssimato uno zero di una funzione polinomiale data. La formula ricorsiva del metodo delle secanti è la seguente:

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \cdot f(x_n)$$

Dove x_nè l'approssimazione dello zero al passo n, x_{n+1}è l'approssimazione dello zero al passo $n + 1$, $f(x_n)$ è il valore della funzione polinomiale in x_n e $f(x_{n-1})$ è il valore della funzione polinomiale in x_{n-1}.

La ricorsione deve terminare quando si verifica una delle seguenti condizioni:

1. l'approssimazione corrente della soluzione (x_{n+1}) è sufficientemente buona, ovvero dato un parametro di tolleranza di approssimazione `t`, si verifica la condizione

$$|x_{n+1} - x_n| \leq t$$

2. il numero di iterazioni effettuate è uguale al numero massimo di iterazioni ammesse `max_iter`. Se il parametro `max_iter` è minore o uguale a `0` vale solo la condizione di uscita 1.

La funzione prende in input i seguenti parametri:

- un `polinomio p` che rappresenta la funzione polinomiale di cui calcolare uno zero;

- due valori iniziali `x0` e `x1` da cui far partire il procedimento ricorsivo;
- il parametro di tolleranza `t`;
- il numero massimo di iterazioni `max_iter`.

La funzione deve ritornare la soluzione individuata.

Si consiglia di utilizzare una funzione ausiliaria per la risoluzione dell'esercizio. Non saranno considerate valide soluzione che non fanno uso della ricorsione.

Esercizio 2 (punti 9)
Creare i file `prezzototale.h` e `prezzototale.c` che consentano di definire la seguente struttura:

```
typedef struct articolo {
    char nome[11];
    int prezzo;
} articolo;
```

e la funzione:

```
extern void TrovaArticoli(const articolo* a, size_t a_size, int sum);
```

la procedura prende in input un vettore di articoli (`a`), la sua dimensione (`a_size`) e un numero intero `sum` e deve stampare a video (standard output), utilizzando un approccio di backtracking, tutti i possibili gruppi di articoli il cui valore totale è esattamente uguale a `sum`. L'output dovrà avere il seguente formato:

```
nome_articolo_1, nome_articolo_2,
nome_articolo_2, nome_articolo_3,
nome_articolo_4
```

ogni riga dovrà contenere una soluzione, ovvero i nomi degli articoli la cui somma dei prezzi vale esattamente `sum`. I nomi degli articoli dovranno essere separati da virgola.

Se il problema non ammette soluzioni oppure se il vettore di articoli è vuoto la funzione non deve stampare nulla. In output non devono comparire soluzioni ripetute. Ad esempio, un output del tipo:

```
nome_articolo_1, nome_articolo_2,
nome_articolo_2, nome_articolo_1,
```

è da considerarsi sbagliato in quanto la stessa soluzione compare due volte. In ogni caso non importa l'ordine con cui vengono stampate a video le soluzioni.

Si noti che per risolvere il problema potrebbe essere necessario utilizzare una funzione ausiliaria.
N.B. Non sarà considerata valida nessuna soluzione che non faccia uso del backtracking.

Esempio:
input:

```
sum = 50
articoli = {{Monopoli, 20}, {Carcassone, 30}, {Perudo, 20}}
```

output:

```
Carcassone, Perudo,
Monopoli, Carcassone,
```

Esercizio 3 (punti 5)
Nel file `freetree.c` definire la funzione corrispondente alla seguente dichiarazione:

```
extern void FreeTree(tree t);
```

La funzione prende in input un albero di interi `t` e deve attraversarlo in pre-ordine stampando a video (standard output) i suoi elementi (numeri interi) e liberando la memoria occupata dai suoi nodi. Il formato dell'output dovrà essere il seguente:

```
{element1}, {element2}, {element3},
```

Se l'albero è vuoto la funzione non deve stampare nulla.

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/20190609_esame/tree_int/tree__int_8h.html

Esercizio 4 (punti 6)
Sia data la seguente definizione:

```
typedef struct coords {
	float x;
	float 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 decimali (in formato ASCII) 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 {
       float x;
       float 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/20190609_esame/list_coords/list__coords_8h.html

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

```
extern tree CutTree(tree t, int cut_value);
```

La funzione prende in input un albero di interi `t` e un numero intero `cut_value`. La funzione deve scorrere l'albero ed eliminare, liberando la memoria e aggiornando opportunamente i puntatori, tutti i sottoalberi aventi radice uguale a `cut_value`. Dato ad esempio l'albero in Figura 1 e un `cut_value` uguale a 4, la funzione deve ritornare l'albero in Figura 2. Se la radice dell'albero ha chiave `cut_value` o se l'albero `t` è vuoto la funzione deve ritornare un albero vuoto. Se l'albero `t` non contiene nessun nodo con chiave `cut_value` la funzione deve ritornare l'albero originale.

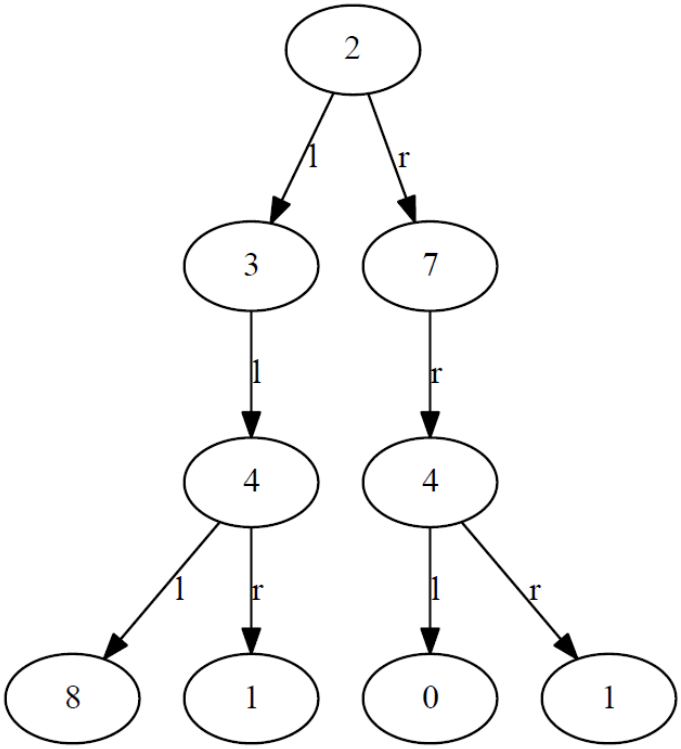

Figura 1

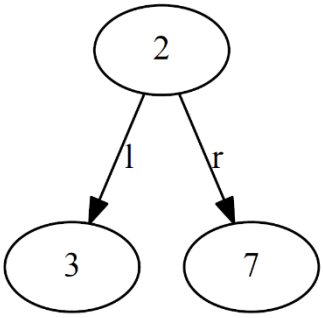

Figura 2

Per la risoluzione di questo esercizio avete a disposizione le definizioni e le primitive fornite per l'esercizio 3.