Note

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

Esercizio 1 (5 punti)

Nel file `duplicate.c` implementare la definizione della funzione:

```
extern bool cifre_duplicate(unsigned long long n);
```

La funzione prende come input un numero intero senza segno (`long long`) e verifica se nella sua rappresentazione in base 10 una cifra compare più di una volta. Ad esempio il numero 123 non ha cifre duplicate (la funzione deve ritornare `false`), perché 1 compare una sola volta, 2 compare una sola volta e 3 compare una sola volta. Invece 1231 ha cifre duplicate (la funzione deve ritornare `true`), perché 1 compare due volte.

Esercizio 2 (6 punti)

Nel file `clamped.c` implementare la definizione della funzione:

```
extern int *clamped(const int *v, size_t n, int min, int max);
```

La funzione prende come input un puntatore ad un vettore di numeri interi con segno di dimensione n e due valori min e max. In output restituisce un vettore allocato dinamicamente della stessa dimensione, in cui ogni valore viene limitato secondo questa regola:

$$y = \begin{cases} min & \text{se } x < min \\ x & \text{se } min \leq x \leq max \\ max & \text{se } x > max \end{cases}$$

Se il puntatore è NULL, se la dimensione del vettore è 0 o se il limite minimo è maggiore del valore massimo, la funzione ritorna NULL.

Esercizio 3 (7 punti)

Creare i file `matrix.h` e `matrix.c` che consentano di utilizzare le seguenti enumerazioni e strutture:

```
struct matrix {
       size_t rows, cols;
       double *data;
};
```

```
struct bmatrix {
       size_t rows, cols;
       bool *data;
};
```

```
enum comparisons {
       LT, LE, EQ,
       NE, GE, GT
};
```

e la funzione:

```
extern struct bmatrix *mat_boolean(const struct matrix *m, double rhs, enum comparisons
cmp);
```

Le struct consentono di rappresentare matrici di dimensioni arbitraria, dove `rows` è il numero di righe, `cols` è il numero di colonne e `data` è un puntatore a `rows×cols` valori di tipo `double` o `bool` memorizzati per righe. L'enum comparisons elenca tutti i possibili confronti che si possono fare tra valori numerici: LT = lower than = minore, LE = lower or equal = minore o uguale, EQ = equal = uguale, NE = not equal = diverso, GE = greater or equal = maggiore o uguale, GT = greater than = maggiore.

La funzione accetta come parametri un puntatore ad una matrice `m`, un valore numerico `rhs` e un confronto `cmp` da effettuare tra tutti gli elementi della matrice e il valore fornito. La funzione deve restituire un puntatore a una nuova bmatrix allocata dinamicamente con il risultato del confronto. Ad esempio, data la matrice

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

chiamare la funzione `mat_boolean(A, 5, GE)` restituisce la matrice

$$\begin{pmatrix} false & false & false \\ false & true & true \\ true & true & true \end{pmatrix}$$

ovvero la matrice ottenuta facendo $1 \geq 5, 2 \geq 5, 3 \geq 5$, e così via.
Se il puntatore alla matrice di input è NULL, la funzione restituisce NULL.

Esercizio 4 (7 punti)

Creare i file `stringhe.h` e `stringhe.c` che consentano di utilizzare la seguente funzione:

```
extern char *title(const char *str);
```

La funzione riceve in input un puntatore a una stringa C `str` e in uscita deve produrre una nuova stringa C allocata dinamicamente che contenga la stringa `str` con l'iniziale di ogni parola in maiuscolo. Con "parola" si intende qualsiasi sequenza che non contenga whitespace (spazi, tabulazioni o a capo).
La funzione deve gestire correttamente qualsiasi carattere ASCII nelle parole e se il puntatore è NULL, deve ritornare NULL.

Esercizio 5 (8 punti)

Creare i file `tirocini.h` e `tirocini.c` che consentano di utilizzare la seguente struttura:

```
struct tirocinio {
    char *nome;
    char *azienda1;
    char *azienda2;
    char *azienda3;
};
```

e la funzione:

```
extern bool tirocinio_load(FILE *f, struct tirocinio *t);
```

La struct consente di rappresentare le preferenze di tirocinio di uno studente, il cui nome è contenuto nella stringa C puntata da nome, assieme ai nomi delle tre aziende preferite. Nella struct sono contenuti puntatori a stringhe allocate dinamicamente (su heap).

La funzione accetta come parametri un puntatore a file già aperto in lettura in modalità tradotta (testo) e un puntatore ad una struct tirocinio già allocata. Non sono invece allocati, né azzerati, i puntatori nome, azienda1, azienda2 e azienda3 del tirocinio.
La funzione deve leggere **una sola riga** da un file di testo con questa struttura:

```
Francesco,Edera,Tetrapak,System↲
Marco Bianchi,System,Doxee,Edera↲
Matteo Rossi,Comune di Modena,Tetrapak,System↲
Luca Neri,Foodpartner,EFFER,NonStop Recruitment↲
Giovanni Verdi,Erre Technology Group,AMARIS Group,UniCredit↲
```

e riempire opportunamente la struttura di cui ha ricevuto l'indirizzo. Il file consiste di righe terminate da un a capo (anche l'ultima), in cui è presente il nome dello studente, seguito dal nome di tre aziende. Questi nomi sono separati da virgole. Supponiamo che il file sia corretto (non serve quindi fare accurati controlli di errore). I nomi degli studenti o delle aziende possono contenere spazi.

La funzione deve ritornare true se riesce a leggere correttamente una riga del file, false altrimenti. Questo dovrebbe accadere solo alla fine del file. La funzione non deve né aprire, né chiudere il file.