Nota

È considerato errore qualsiasi output non richiesto dagli esercizi.

Esercizio 1 (punti 6)

Creare il file `conta.c` che consenta di utilizzare la seguente funzione:

```
extern unsigned int conta_occorrenze(const char *testo, const char *stringa);
```

La funzione accetta due stringhe zero terminate. Deve restituire il numero di occorrenze di stringa in testo. Ad esempio, dopo la chiamata della funzione:

```
i = conta_occorrenze("Qui bisogna cercare la parola cercare","cercare");
```

i varrà 2. La funzione deve ritornare 0 se testo o stringa sono NULL o contengono solo il terminatore (sono vuote).

Esercizio 2 (punti 6)

La notazione $n!!$ denota il **semifattoriale** (o *doppio fattoriale*) di $n \in \mathbb{N}$, definito come

$$n!! = \begin{cases} 1 & se\ n = 0\ o\ n = 1; \\ n \cdot (n-2)!! & se\ n \geq 2. \end{cases}$$

per esempio $8!! = 8 \cdot 6 \cdot 4 \cdot 2 = 384$ e $9!! = 9 \cdot 7 \cdot 5 \cdot 3 = 945$.
Nel file `matematica.c` implementare in linguaggio C la funzione corrispondente alla seguente dichiarazione:

```
extern double semifattoriale(char n);
```

La funzione deve restituire il semifattoriale di `n`. Se `n` è negativo, deve restituire `-1`. Non è consentito l'uso di librerie esterne. Utilizzare internamente double per tutti i calcoli, per avere una precisione sufficiente.

Esercizio 3 (punti 7)

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

```
extern char *fgets_malloc(FILE *f);
```

La funzione accetta un puntatore a `FILE` aperto in lettura in modalità tradotta (testo) e deve leggere tutti i caratteri fino al primo a capo o fino alla fine del file. La funzione ritorna una stringa zero terminata allocata dinamicamente nell'heap, contenente i caratteri letti.
Il carattere a capo, se presente, non deve essere inserito nella stringa ritornata.
Se quindi viene letta una riga contenente solo il carattere a capo, la funzione ritorna un puntatore ad un'area di memoria grande 1 byte contenente solo il terminatore (una stringa con zero caratteri).
Se invece la funzione non riesce a leggere nessun carattere (la prima lettura ritorna EOF), la funzione ritorna NULL, non allocando quindi nulla. Questo segnala che la lettura è fallita.

Non è possibile utilizzare rewind in questa funzione (perché non è detto che venga eseguita a partire dall'inizio del file), ne fare assunzioni sulla massima dimensione della stringa letta.

Esercizio 4 (punti 6)

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

```
struct sample{
	int idSample;
	char nomeCategoria[20];
	double accuracy;
};
```

e le funzioni:

```
extern int sample_scrivi(FILE* f, const struct sample* s);
extern int sample_leggi(FILE* f, struct sample* s);
```

Entrambe le funzioni lavorano con file binari in cui ogni sample è salvato come:
1) un intero a 32 bit in little endian che contiene l'idSample
2) un array di 20 byte contenenti il nomeCategoria come stringa zero terminata (al massimo avrà 19 caratteri)
3) un numero in virgola mobile a 64 bit codificato secondo il formato IEEE 754 contenente l'accuracy.

Entrambe le funzioni accettano un puntatore a `FILE` aperto in scrittura/lettura in modalità non tradotta (binaria) e un puntatore ad un sample da scrivere o da leggere.
Le funzioni ritornano 1 se sono riuscite a scrivere o leggere un intero sample correttamente, 0 altrimenti. In particolare sample_leggi ritorna 0 se raggiunge la fine del file prima di aver letto interamente il sample (viene utilizzato per sapere se nel file non ci sono più sample).

Esercizio 5 (punti 7)

Una matrice quadrata $A = \left(a_j^i\right) \in \mathcal{M}_n(\mathbb{K})$ si dice *triangolare alta* se:

$$\left(a_j^i \neq 0\right) \Rightarrow (i \leq j).$$

In altri termini, A è triangolare alta se tutti gli elementi "al di sotto" della sua diagonale principale sono nulli.
Creare i file `matrix.h` e `matrix.c` che consentano di utilizzare la seguente struttura:

```
struct matrix {
	size_t N,M;
	double *data;
};
```

e la funzione:

```
extern int mat_isupper(const struct matrix *matr);
```

La struct consente di rappresentare matrici di dimensioni arbitraria, dove `N` è il numero di righe, `M` è il numero di colonne e `data` è un puntatore a `N×M` valori di tipo `double` memorizzati per righe. Consideriamo ad esempio la matrice

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

questo corrisponderebbe ad una variabile `struct matrix A`, con `A.N = 2`, `A.M = 3` e `A.data` che punta ad un area di memoria contenente i valori `{ 1.0, 2.0, 3.0, 4.0, 5.0, 6.0 }`.
La funzione accetta come parametro un puntatore ad una matrice e deve ritornare 1 se la matrice è quadrata e triangolare alta, 0 altrimenti. Il puntatore non sarà mai `NULL`.