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 (punti 5)

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

```
extern bool cifre_duplicate_oct(unsigned int n);
```

La funzione prende come input un numero intero senza segno (`unsigned int`) e verifica se **nella sua rappresentazione in base 8** una cifra compare più di una volta. Ad esempio il numero 127 non ha cifre duplicate (la funzione deve ritornare `false`), perché 1 compare una sola volta, 2 compare una sola volta e 7 compare una sola volta. Invece 4234 ha cifre duplicate (la funzione deve ritornare `true`), perché 4 compare due volte.

Esercizio 2 (punti 6)

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

```
extern int *rimuovi_multipli(const int *v, size_t size, size_t *newsize);
```

La funzione accetta un puntatore ad un vettore di size numeri interi (è garantito che non saranno minori di zero o nulli) e deve creare e ritornare un nuovo vettore allocato dinamicamente su heap che contenga solo i numeri che non sono multipli di altri elementi del vettore. Nella variabile puntata da newsize dovrà inserire il numero di elementi del vettore ritornato.
Ad esempio, dato il vettore di 5 elementi { 2, 3, 4, 5, 6 }, dovrà ritornare { 2, 3, 5 }. Se invece il vettore fosse { 12, 10, 4, 8, 5 }, dovrebbe ritornare { 4, 5 }.

Esercizio 3 (punti 7)

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

```
extern char *spacefill(const char *str, size_t n);
```

La funzione riceve in input un puntatore a una stringa C `str` e una dimensione `n` e in uscita deve produrre una nuova stringa C, di almeno `n` caratteri, allocata dinamicamente su heap, che contenga la stringa `str` con davanti tanti caratteri spazio. Se str è già lunga `n` o più lunga, non deve aggiungere nulla.
Ad esempio invocando la funzione così: `spacefill("ciao", 10)` otterremmo in output `"      ciao"`, mentre invocando la funzione così: `spacefill("ciao", 2)` otterremmo in output `"ciao"`
La funzione deve gestire correttamente qualsiasi carattere ASCII e se il puntatore è NULL, deve ritornare NULL.

Esercizio 4 (7 punti)

Creare i file `shapes.h` e `shapes.c` che consentano di utilizzare la seguente struct in grado di rappresentare linee in uno spazio 2D, associate ad uno spessore:

```
struct line {
    int16_t x1, y1;
    int16_t x2, y2;
    uint8_t thickness;
};
```

e la funzione:

```
extern bool line_load(FILE *f, struct line *pln);
```

La funzione riceve in input un puntatore ad un file già aperto in modalità lettura non tradotta (binario) e **legge una sola linea** memorizzata su file con 4 valori a 16 bit con segno in little endian e un intero a 8 bit senza segno. I dati letti vanno inseriti nella `struct line` puntata da `pln`. Il puntatore passato sarà sempre valido e già allocato. Al termine della lettura la posizione nel file sarà di 9 byte più avanti e pronta per leggere i dati successivi (ad esempio della prossima linea). La funzione ritorna true se è riuscita a leggere i 5 campi correttamente, false altrimenti.

Ad esempio il file seguente (visto come in un editor esadecimale)

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  00 00 00 00 0A 00 05 00 09 05 00 05 00 14 00 0F  ................
00000010  00 0A                                           ..
```

contiene due linee [x1=0, y1=0, x2=10, y2=5, thickness=9] e [x1=5, y1=5, x2=20, y2=15, thickness=10].

Esercizio 5 (8 punti)

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

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

e la funzione:

```
extern struct matrix *mat_permute_rows(const struct matrix *m, const size_t *p);
```

La struct consente 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` 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.rows = 2`, `A.cols = 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 parametri un puntatore ad una matrice `m` e un puntatore ad un vettore di indici `p` e deve restituire un puntatore a una nuova matrice allocata dinamicamente. La matrice è ottenuta mettendo nella riga i-esima la riga della matrice originale il cui indice è specificato alla posizione i-esima del vettore p.

Ad esempio, data la matrice

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

e il vettore di indici

$$\begin{pmatrix} 2 & 1 & 0 \end{pmatrix}$$

la funzione restituisce la matrice

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

ovvero la matrice ottenuta mettendo una sotto l'altra le righe 2, 1 e 0 della matrice originale.
I puntatori e tutti i dati di input saranno sempre validi, quindi non serve fare alcun controllo.