Nota

È considerato errore qualsiasi output non richiesto dagli esercizi.

Esercizio 1 (punti 5)

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

```
extern double media (double *arr, size_t n);
```

La funzione accetta come parametri un puntatore ad un vettore di double `arr` e un dato di tipo `size_t` che ne indica la dimensione e deve restituire la media dei valori contenuti nell'array.

Esercizio 2 (punti 7)

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

```
extern size_t conta_parole (const char *s);
```

La funzione accetta come parametro una stringa zero terminata e deve restituire in un dato di tipo `size_t` quante parole sono presenti all'interno della stringa, dove con "parola" intendiamo una sequenza di caratteri diversi da spazio.

Esercizio 3 (punti 6)

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

```
extern void stampa_cono (unsigned int h);
```

La funzione deve inviare a `stdout` un cono composto di due caratteri '_' alla base, caratteri '\' e '/' sulle diagonali e un altro '_' sulla punta. Il parametro `h` (che sarà sempre maggiore di zero) regola l'altezza del cono, ovvero il numero di coppie di barre diagonali (un cono occuperà `h+1` righe). Ad esempio chiamando la funzione con `h=1`, la funzione deve inviare su stdout:

```
 _
_/ \_
```

Chiamando la funzione con `h=3`, la funzione deve inviare su stdout:

```
   _
  / \
 /   \
_/     \_
```

Ovvero (visualizzando ogni carattere in una cella della seguente tabella):

<sp.>	<sp.>	<sp.>	<sp.>	_	<sp.>	<sp.>	<sp.>	<sp.>	<a capo>
<sp.>	<sp.>	<sp.>	/	<sp.>	\	<sp.>	<sp.>	<sp.>	<a capo>
<sp.>	<sp.>	/	<sp.>	<sp.>	<sp.>	\	<sp.>	<sp.>	<a capo>
_	/	<sp.>	<sp.>	<sp.>	<sp.>	<sp.>	\	_	<a capo>

Si ricorda che in C il carattere `\` deve essere inserito come `'\\'`.

Esercizio 4 (punti 8)

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

```
struct stringa {
	unsigned char length;
	char *s;
};
```

e la funzione:

```
extern struct stringa *read_stringhe_bin (const char *filename, unsigned int *pn);
```

La struttura contiene il campo `length` che contiene la lunghezza della stringa (eventualmente 0) e il campo `s` che punta ad una stringa zero terminata (di lunghezza length).
La funzione accetta come parametro un nome di file che deve essere aperto in lettura in modalità non tradotta (binario) e un puntatore ad una variabile di tipo `unsigned int` in cui si dovrà inserire il numero di stringhe presenti nel file. Il file è composto di una sequenza di elementi di lunghezza variabile in cui un byte indica la lunghezza `n` della stringa e di seguito ci sono `n` byte contenenti i caratteri della stringa.
La funzione deve ritornare un puntatore ad una nuova zona di memoria (allocata dinamicamente nell'heap) contenente tutte le stringhe lette dal file. Il numero di stringhe non è noto a priori e non può essere vincolato dal codice. Anche l'elemento s di stringa deve essere allocato dinamicamente nell'heap.

Ad esempio, un file valido (mostrato come in un editor esadecimale) è:

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  05 43 69 61 6F 21 00 03 61 62 63 0E 50 72 6F 67  .Ciao!..abc.Prog
00000010  72 61 6D 6D 61 7A 69 6F 6E 65                    rammazione
```

Il file contiene una stringa di lunghezza 5 ("Ciao!"), una stringa di lunghezza 0 (""), una stringa di lunghezza 3 ("abc") e una stringa di lunghezza 14 ("Programmazione"). In questo caso la funzione dovrà impostare la variabile puntata da `pn` a 4. Per testare la funzione, utilizzare i file `stringhe1.bin` e `stringhe2.bin` disponibili nella pagina dell'esame.

Esercizio 5 (punti 7)

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

```
extern double *diag (double *matr, size_t n);
```

La funzione accetta come parametro `matr`, un puntatore ad una zona di memoria contenente una matrice quadrata di lato `n` (il secondo parametro), memorizzata per righe, ovvero contenente `n*n` elementi dei quali i primi `n` sono la prima riga, i successivi `n` la seconda e così via.
La funzione deve ritornare un puntatore ad una nuova zona di memoria (allocata dinamicamente nell'heap) contenente gli elementi della diagonale principale di `matr`.

Ad esempio, se `matr` puntasse alla matrice:

1	2	3
4	5	6
7	8	9

(con `n=3`) ovvero ad una zona di memoria contenente i valori 1, 2, 3, 4, 5, 6, 7, 8, 9, la funzione deve ritornare un puntatore ad una zona di memoria grande 3 double contenenti i valori 1, 5, 9.
La dimensione della matrice non può essere vincolata nel codice.