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 *array_somma (const double *arr1, const double *arr2, size_t n);
```

La funzione accetta come parametri due puntatori a vettori di double `arr1` e `arr2` e un dato di tipo `size_t` che ne indica la dimensione `n` (gli array sono grandi uguali). La funzione deve restituire un puntatore ad un vettore allocato dinamicamente nell'heap, formato da `n` elementi di tipo double calcolati come la somma dei corrispondenti elementi di `arr1` e `arr2`. `arr1` e `arr2` non possono essere `NULL` e `n` sarà sempre maggiore di 0.

Esercizio 2 (punti 8)

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

```
extern char *cerca_primo (char *s, const char *list);
```

La funzione accetta come parametro una stringa zero terminata in cui cercare caratteri e una sequenza di caratteri (anch'essa zero terminata) e deve restituire un puntatore alla prima occorrenza in `s` di un qualsiasi carattere presente in `list`. Ad esempio se cercassimo in `s="aereo"` un carattere nella lista `list="fyr"` dovrebbe ritornare un puntatore al secondo carattere della stringa `s`. Nel caso il carattere non sia presente, oppure se `s` o `list` sono vuote, deve ritornare `NULL`. `s` e `list` non sono mai putatori `NULL`.

Esercizio 3 (punti 6)

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

```
extern void stampa_cornicetta (const char *s);
```

La funzione deve inviare a `stdout` la stringa passata come parametro circondandola con una cornicetta composta dei caratteri '\' e '/' agli spigoli e di '-' e '|' sui lati. Prima e dopo la stringa bisogna inserire uno spazio. Ad esempio chiamando la funzione con `s="ciao"`, la funzione deve inviare su stdout:

```
/------\
| ciao |
\------/
```

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

/	-	-	-	-	-	-	\	<a capo>
|	<sp.>	c	i	a	o	<sp.>	|	<a capo>
\	-	-	-	-	-	-	/	<a capo>

Si ricorda che in C il carattere '\' deve essere inserito come '\\'. Gli <a capo> a fine riga sono obbligatori per una soluzione corretta.

Esercizio 4 (punti 8)

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

```
struct complesso {
       double re,im;
};
```

e le funzioni:

```
extern int read_complesso (struct complesso *comp, FILE *f);
extern void write_complesso (const struct complesso *comp, FILE *f);
extern void prodotto_complesso (struct complesso *comp1, const struct complesso *comp2);
```

La struct consente di rappresentare numeri complessi come coppia ordinata di valori reali, ovvero la parte reale e la parte immaginaria.
Questi numeri vengono rappresentati in base dieci come sequenza di caratteri, con parte reale e immaginaria separate da spazi e seguite da un a capo. Ad esempio i due valori complessi $2 + 4i$ e $1 - i$ verrebbero rappresentati su file come:

```
2.000000 4.000000
1.000000 -1.000000
```

La funzione `read_complesso` accetta come parametro un puntatore a un numero complesso `comp` e un puntatore a `FILE` aperto in lettura in modalità tradotta (testo). La funzione deve leggere dal file i due campi del numero complesso e ritornare 1 se è riuscita a leggere 2 valori, 0 altrimenti.
La funzione `write_complesso` effettua l'operazione inversa, ovvero scrive sul file `f` i due campi del numero complesso separandoli con uno spazio e andando a capo.
La funzione `prodotto_complesso` esegue il prodotto dei due valori `comp1` e `comp2` e mette il risultato in `comp1`. Si ricorda che il prodotto di numeri complessi si esegue così:

$$(a + ib)(c + id) = (ac - bd) + i(ad + bc)$$

Sul sito viene fornito il file `complessi.txt` che potete usare per fare delle prove.

Esercizio 5 (punti 6)

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

```
extern double det3x3 (double *matr);
```

La funzione accetta come parametro `matr`, un puntatore ad una zona di memoria contenente una matrice quadrata di lato 3, memorizzata per righe, ovvero contenente 3x3 elementi dei quali i primi 3 sono la prima riga, i successivi 3 la seconda e gli ultimi tre la terza. La funzione deve ritornare il determinante della matrice passata come parametro. Si ricorda che:

$$\det\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = aei + bfg + cdh - gec - hfa - idb$$

Ad esempio, se `matr` puntasse alla matrice:

1	2	3
1	1	1
1	2	1

ovvero ad una zona di memoria contenente i valori 1, 2, 3, 1, 1, 1, 1, 2, 1, la funzione deve ritornare il valore 2.