Nota

È considerato errore qualsiasi output non richiesto dagli esercizi.

Esercizio 1 (punti 7)

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

```
extern double *array_remove (const double *arr, size_t n, size_t pos);
```

La funzione accetta come parametri un puntatore a un vettore di double `arr`, un dato di tipo `size_t` che ne indica la dimensione `n`, una posizione `pos`.
Se `pos` indica una posizione all'interno dell'array, ovvero se è minore di `n`, la funzione deve ritornare un puntatore ad una nuova zona di memoria (allocata dinamicamente nell'heap) contenente una copia dell'array senza l'elemento alla posizione `pos`. Se `pos` è maggiore o uguale a `n` o se `arr` è `NULL`, la funzione deve ritornare NULL.

Esercizio 2 (punti 7)

Nel file `trigonometria.c` implementare in linguaggio C la funzione corrispondente alla seguente dichiarazione:

```
extern double seno(double x);
```

La funzione deve calcolare il valore di $\text{sen}(x)$ utilizzando il seguente sviluppo in serie di Taylor:

$$\text{sen}(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

Non è consentito l'uso di librerie esterne.

Esercizio 3 (punti 6)

Creare il file `encrypt.c` che contenga la definizione della seguente funzione:

```
extern void encrypt(char *s, unsigned int n);
```

La funzione accetta una sequenza `s` di `n` char e la codifica sostituendo ad ogni char il suo valore trasformato con uno XOR bit a bit con il valore esadecimale AA. Per le proprietà dello XOR, l'operazione è invertibile, quindi riapplicando la funzione sulla sequenza codificata si riottiene quella originale.

Esercizio 4 (punti 5)

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

```
extern void write_bin(const double *values, size_t n, FILE *f);
```

La funzione accetta come parametro `values`, un puntatore ad una zona di memoria contenente un vettore di double, un dato di tipo `size_t` che ne indica la dimensione `n` e un puntatore a un file aperto in modalità non tradotta (binario).
La funzione deve scrivere sul file ogni valore in formato binario come questo è rappresentato in memoria. Se `values` è `NULL` o `n=0`, la funzione non invia nulla in output.

Esercizio 5 (punti 8)

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 matrix_read(struct matrix *matr, FILE *f);
```

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 deve leggere la matrice `matr` dal file `f` (già aperto). Questa è memorizzata in formato testuale decimale conil valore di N seguito da <a capo>, il valore di M seguito da <a capo>, i valori della matrice separati da <tabulazione> (\t in linguaggio C) all'interno della riga e con un <a capo> alla fine di ogni riga (compresa l'ultima).

Facendo riferimento all'esempio precedente, A sarebbe scritta sul file come

```
2↵
3↵
1.000000	→	2.000000	→	3.000000↵
4.000000	→	5.000000	→	6.000000↵
```

Visualizzando in una tabella i singoli caratteri si vedrebbe

2	\n																							
3	\n																							
1	.	0	0	0	0	0	0	\t	2	.	0	0	0	0	0	0	\t	3	.	0	0	0	0	0	0	\n
4	.	0	0	0	0	0	0	\t	5	.	0	0	0	0	0	0	\t	6	.	0	0	0	0	0	0	\n

La funzione ritorna 1 se la lettura è andata a buon fine, 0 se per qualche motivo è fallita.