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 (5 punti)

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

```
extern int *prodotto(const int *v, size_t n);
```

La funzione prende come input un puntatore a un vettore di `int` di dimensione `n` e deve restituire un nuovo vettore allocato dinamicamente su heap, in cui l'elemento di indice `i` è ottenuto come il prodotto di tutti quelli di indice `j≠i`.

Ad esempio dato il vettore

```
1 2 3 4 5
```

la funzione deve restitire il vettore

```
120 60 40 30 24
```

Se `v` è `NULL` o se contiene `0` o `1` elementi, la funzione deve ritornare `NULL`.

Esercizio 2 (6 punti)

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

```
extern void elimina_consecutivi(char *str);
```

La funzione riceve in input un puntatore a una stringa C `str` e deve modificarla sostituendo ad ogni sequenza di caratteri uguali e consecutivi una sola occorrenza del carattere ripetuto. Ad esempio data la stringa

```
cccaaaneeee
```

deve modificare `str` in

```
cane
```

Oppure, data la stringa

```
mamma
```

deve produrre

```
mama
```

Se `str` è `NULL`, non deve fare nulla.

Esercizio 3 (7 punti)

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

```
extern void stampa_scomposizione(unsigned int n);
```

La funzione prende come input un numero intero senza segno `n` e deve scrivere su `stdout` la sua scomposizione in fattori primi nel seguente modo:

- i fattori primi vengono elencati dal più piccolo al più grande
- ogni fattore primo deve essere separato dagli altri dai tre caratteri `" * "` (spazio, asterisco, spazio)
- se il fattore è presente più di una volta, dopo il numero si inserisce un carattere `^` (apice) seguito dall'esponente con cui compare nel numero
- convenzionalmente, se `n=0` l'output sarà `0`, se `n=1` l'output sarà `1`.

Ad esempio, se `n=120`, l'output dovrebbe essere:

```
2^3 * 3 * 5
```

Esercizio 4 (7 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_delete_row(const struct matrix *m, size_t i);
```

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 indice di riga `i` e deve restituire un puntatore a una nuova matrice allocata dinamicamente. La matrice è ottenuta copiando i dati di quella di

input ad eccezione della riga i-esima, con `i=0` per la prima riga. Se `i` specifica una riga non valida, se `m` ha una sola riga, o se `m` è `NULL`, la funzione restituisce `NULL`.

Ad esempio, data la matrice

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

la funzione, chiamata con il parametro `i` uguale a `1`, restituisce la nuova matrice

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

Esercizio 5 (8 punti)

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

```
struct vec_double {
    uint32_t size;
    double *data;
};
```

e la funzione:

```
extern struct vec_double *read_vec_double(const char *filename);
```

La struct consente di rappresentare vettori di `double` di lunghezza arbitraria, dove `size` contiene il numero di elementi e `data` punta ad un area di memoria di dimensione opportuna.

La funzione accetta come parametri una stringa C con il nome di un file e deve restituire un puntatore a una `struct vec_double`, allocata dinamicamente su heap, contenente i dati letti da file.

Il formato del file è composto da un numero intero senza segno a 32 bit che indica il numero di elementi nel vettore, seguito da una sequenza di double a 64 bit senza separatori di alcun tipo. Tutti i valori sono memorizzati in little endian (come nei processori Intel). La funzione deve aprire il file in modalità non tradotta (binaria), leggere i dati del file allocando dinamicamente memoria per contenere i valori e impostare opportunamente i campi della struct. La funzione ritorna `NULL` se non è possibile aprire il file, leggere il numero di elementi, o leggere tanti elementi quanti indicati dal primo campo.

Ad esempio, dato 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  03 00 00 00 9A 99 99 99 99 99 B9 3F 00 00 00 00  ....š™™™™™¹?....
00000010  00 00 E0 3F 00 00 00 00 00 00 E4 3F              ..à?......ä?
```

la funzione ritornerebbe un puntatore a struct con 3 nel campo `size` e un puntatore ai 3 double letti in `data`:

```
{ 0.1 , 0.5 , 0.625 }
```