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 `array_rotate.c` implementare la definizione della funzione:

```
extern int *rotate(const int *vec, size_t len, size_t r);
```

La funzione prende come input un puntatore a un vettore di `int`, la sua lunghezza e un valore `r`. La funzione deve restituire un puntatore ad un nuovo vettore allocato dinamicamente nell'heap con gli stessi elementi del vettore in ingresso, ma ruotati in avanti di `r` posizioni, ovvero l'elemento in posizione n viene messo in posizione n+r. Se n+r è oltre la fine del vettore, si ricomincia da 0.
Ad esempio il vettore { 1, 2, 3, 4 }:
ruotato di 1 diventa { 4, 1, 2, 3 }
ruotato di 2 diventa { 3, 4, 1, 2 }
ruotato di 3 diventa { 2, 3, 4, 1 }
ruotato di 4 diventa { 1, 2, 3, 4 }
ruotato di 5 diventa { 4, 1, 2, 3 }
…

Esercizio 2 (7 punti)

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

```
extern bool is_gif(const char *filename);
```

La funzione accetta in input una stringa C che contiene il nome di un file e verifica se il file contiene un immagine in formato GIF, verificando che i primi 6 byte contengano i caratteri `GIF89a` oppure `GIF87a`. Se il file non esiste, se non è possibile leggere 6 byte o se i 6 byte letti non corrispondono ad una delle due possibilità, la funzione ritorna `false`, altrimenti `true`.

Suggerimento: utilizzate HxD per aprire gli esempi forniti e verificare quali file sono effettivamente GIF.

Esercizio 3 (7 punti)

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

```
extern int *convolution3(const int *v, size_t lenv, const int k[3]);
```

La funzione accetta come parametri un puntatore ad un vettore v di int e la relativa lunghezza, un puntatore ad un kernel (un altro vettore di int) di lunghezza 3. La funzione deve restituire un puntatore ad un nuovo vettore allocato dinamicamente nell'heap con lo stesso numero di elementi del vettore v che contenga la convoluzione tra v e k. Se v o k sono NULL la funzione ritorna NULL.

L'operazione di convoluzione tra un vettore e un altro di lunghezza 3 produce un nuovo vettore i cui elementi possono essere calcolati come:

$$c[n] = (v * k)[n] = \sum_{m=0}^{2} v[n+1-m] \cdot k[m]$$

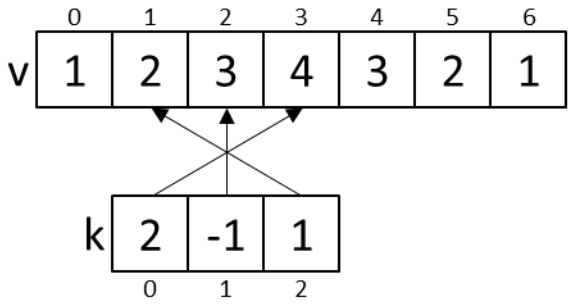
con $0 \leq n < \text{lenv}$. L'operazione assume che

$$v[i] = \begin{cases} v[i] & 0 \leq i < \text{lenv} \\ 0 & altrimenti \end{cases}$$

ovvero che il vettore sia bordato con zeri (padding).

Esempio:
v = { 1, 2, 3, 4, 3, 2, 1 }
k = { 2, -1, 1 }

L'elemento 2 della convoluzione può essere calcolato così:



$$c[2] = v[2+1-0] \cdot k[0] + v[2+1-1] \cdot k[1] + v[2+1-2] \cdot k[2] =$$
$$= v[3] \cdot k[0] + v[2] \cdot k[1] + v[1] \cdot k[2] =$$
$$= 4 \cdot 2 + 3 \cdot (-1) + 2 \cdot 1 =$$
$$= 8 - 3 + 2 = 7$$

L'elemento 0 della convoluzione sarebbe:

$$c[0] = v[0+1-0] \cdot k[0] + v[0+1-1] \cdot k[1] + v[0+1-2] \cdot k[2] =$$
$$= v[1] \cdot k[0] + v[0] \cdot k[1] + v[-1] \cdot k[2] =$$
$$= 2 \cdot 2 + 1 \cdot (-1) + 0 \cdot 1 =$$
$$= 4 - 1 + 0 = 3$$

Notate che se l'indice con cui si accede al vettore esce dallo stesso si assume che ci sia uno 0.

Esercizio 4 (6 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_scale(const struct matrix *m, double x);
```

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 double `x` e deve restituire un puntatore a una nuova matrice allocata dinamicamente che contiene la matrice ottenuta moltiplicando gli elementi della matrice `m` per lo scalare `x`.
Ad esempio:

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

$$\text{mat\_scale}(M, 7.0) \equiv 7 \cdot M = \begin{pmatrix} 7 & 14 & 21 \\ 28 & 35 & 42 \\ 49 & 56 & 63 \end{pmatrix}$$

Se il puntatore passato alla funzione è `NULL` la funzione ritorna `NULL`.

Esercizio 5 (8 punti)

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

```
extern void path_split(const char *str, char **path, char **filename);
```

La funzione prende come parametri una stringa C e due puntatori a puntatore a `char`. La funzione deve dividere la stringa (che sarà un percorso di file) in due parti in corrispondenza del carattere \ più a destra e modificare i puntatori puntati da `path` e `filename` in modo che puntino a due nuove stringhe allocate dinamicamente contenenti le due parti della stringa: la prima fino al carattere \ incluso e la seconda il resto.

Con il percorso `c:\user\esami\FdI2017\esercizi` si otterranno le stringhe `c:\user\esami\FdI2017\` e `esercizi`.
Con il percorso `c:\\user\\esami\\FdI2017\\esercizi\\file.txt` si otterranno le stringhe `c:\\user\\esami\\FdI2017\\esercizi\\` e `file.txt`
Con il percorso `esercizi\\` si otterranno le stringhe `esercizi\\` e la stringa vuota (colo terminatore).
Se la stringa è `NULL` la funzione imposta i puntatori puntati da `path` e `filename` a `NULL`.