# Compito 05/02/2020 (2)

## 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 `triangolare.c` implementare la definizione della funzione:

```
extern bool triangolare(int n);
```

La funzione restituisce `true` se n è un numero triangolare, `false` altrimenti.

Un numero n è triangolare se è possibile disporre n oggetti su una griglia regolare in modo da formare un triangolo equilatero.

I primi numeri triangolari sono 1, 3, 6, 10, 15:

```
*         *           *             *               *
         * *         * *           * *             * *
                    * * *         * * *           * * *
                                 * * * *         * * * *
                                                * * * * *
```

Se n $\leq$ 0, la funzione ritorna `false`.

## Esercizio 2 (6 punti)

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

```
extern char *codifica_morse(const char *testo);
```

La funzione `codifica_morse` accetta come parametro un puntatore ad una stringa C e ne ritorna la versione in codice morse, allocata dinamicamente su heap.

La codifica di ogni lettera dell'alfabeto è rappresentata in tabella:

| Lettera | Codice |
|---|---|
| A | • — |
| B | — • • • |
| C | — • — • |
| D | — • • |
| E | • |

| Lettera | Codice |
|---|---|
| F | • • — • |
| G | — — • |
| H | • • • • |
| I | • • |
| J | • — — — |
| K | — • — |
| L | • — • • |
| M | — — |
| N | — • |
| O | — — — |
| P | • — — • |
| Q | — — • — |
| R | • — • |
| S | • • • |
| T | — |
| U | • • — |
| V | • • • — |
| W | • — — |
| X | — • • — |
| Y | — • — — |
| Z | — — • • |

La stringa `testo` conterrà solamente parole composte da lettere maiuscole, separate da un solo spazio, senza spazi prima o dopo. Usare il carattere `.` (punto) per il punto e il carattere _ (trattino basso) per la linea. Tra il codice di una lettera e quello successivo inserire un carattere `spazio`, e tra due parole inserire i caratteri `" / "` (spazio barra spazio).

Ad esempio, data in input la stringa `"CIAO MAMMA"`, la funzione restituisce

```
"_._. .. ._ ___ / __ ._ __ __ ._"
```

Se `testo` è `NULL`, la funzione restituisce `NULL`.

# Esercizio 3 (7 punti)

Creare i file `rectangle.h` e `rectangle.c` che consentano di utilizzare le seguenti strutture:

```c
struct point {
    int x, y;
};
```

```c
struct rect {
    struct point a, b;
};
```

e la funzione:

```c
extern int find_largest(const struct rect *r, size_t n);
```

La struct `point` consente di rappresentare un punto secondo le sue coordinate cartesiane `x` e `y`.

La struct `rect` consente di rappresentare un rettangolo identificato da una coppia di punti, dove `a` e `b` sono vertici opposti del poligono.

Consideriamo ad esempio il rettangolo:

$$P1(0;2) \quad P2(3;2) \quad P3(3;0) \quad P4(0;0)$$

```
P1--------P2
|         |
|         |
P4--------P3
```

Questo può essere rappresentato con una variabile `struct rect R` con

```
R.a = P1    oppure    R.a = P2    oppure    R.a = P3    oppure    R.a = P4
R.b = P3              R.b = P4              R.b = P1              R.b = P2
```

La funzione `find_largest()` riceve in input un puntatore `r` ad una zona di memoria che contiene `n` elementi di tipo `struct rect`, e restituisce l'indice del rettangolo con area massima.

Se `r` è `NULL` o `n=0`, la funzione restituisce `-1`.

Se l'area massima è condivisa da più rettangoli, la funzione restituisce l'indice di quello con indice minore.

# Esercizio 4 (7 punti)

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

```c
struct matrix {
    size_t rows, cols;
    double *data;
};
```

e la funzione:

```c
extern struct matrix *mat_reset_cross(const struct matrix *m, size_t ir, size_t ic);
```

La `struct matrix` 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 `mat_reset_cross()` accetta come parametro un puntatore ad una matrice `m`, un indice di riga `ir`, ed un indice di colonna `ic`, e deve restituire un puntatore a una nuova matrice allocata dinamicamente. La nuova matrice è ottenuta copiando i dati di quella di input ed azzerando la riga `ir` e la colonna `ic`, ad eccezione dell'elemento `(ir, ic)`.

Se `m` è `NULL`, oppure se `ir` indica una riga non valida, oppure se `ic` indica una colonna non valida, 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 `ir=0` ed il parametro `ic=1` restituisce la nuova matrice

$$\begin{pmatrix} 0 & 2 & 0 \\ 4 & 0 & 6 \\ 7 & 0 & 9 \end{pmatrix}$$

# Esercizio 5 (8 punti)

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

```
struct image {
    char name[255];
    size_t height, width;
};
```

e la funzione:

```
extern struct image *read_images(const char* filename, size_t *n);
```

La `struct image` descrive un'immagine, tramite i campi:

- `name`: nome, rappresentato come stringa C
- `height`: altezza in pixel
- `width`: larghezza in pixel

La funzione `read_images` accetta come parametro un nome di file che deve essere aperto in lettura in modalità tradotta (testo) e un puntatore ad una variabile di tipo `size_t` in cui si dovrà inserire il numero di immagini presenti in un file così strutturato:

```
<name>:<height>:<width><a capo>
<name>:<height>:<width><a capo>
<name>:<height>:<width><a capo>
...
```

La funzione deve ritornare un puntatore ad una nuova zona di memoria (allocata dinamicamente nell'heap) contenente tutte le immagini lette dal file. Il numero di immagini non è noto a priori e non può essere vincolato dal codice.

Un esempio di file valido è:

```
ISIC_0000174.jpg:512:350
ISIC_0031820.jpg:420:1000
ISIC_0000382_downsampled.jpg:240:300
```

In questo caso la funzione dovrà impostare la variabile puntata da `n` a 3.

Se non è possibile aprire il file, o se non è possibile leggere nemmeno una immagine, la funzione ritorna `NULL`.