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 `temperature.c` implementare la definizione della seguente funzione:

```
extern size_t gelate(const char *a, size_t n);
```

La funzione riceve in input un vettore `a` di `n` numeri interi che indicano la temperatura misurata in una sequenza di misurazioni successive e deve ritornare il numero di volte in cui si è verificata una gelata, ovvero in cui la temperatura è passata da un valore positivo ad uno negativo o nullo in due misurazioni successive. La sequenza `{ 4, 3, 3, -1, -4, -6 }` indica una situazione con una gelata, mentre la sequenza `{ -9, -11, -5, -8, -3, -4 }` indica una situazione senza alcuna gelata (la temperatura è costantemente sotto zero.

Esercizio 2 (7 punti)

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

```
#include <stdint.h> // Per avere uint8_t
struct image {
    size_t rows, cols;
    uint8_t *data;
};
```

e la funzione:

```
extern struct image *image_doublesize(const struct image *img);
```

La struct consente di rappresentare immagini a livelli di grigio di dimensioni arbitrarie, dove `rows` è il numero di righe, `cols` è il numero di colonne e `data` è un puntatore a `rows×cols` valori di tipo `uint8_t` memorizzati per righe. Fondamentalmente un'immagine è una matrice di numeri dove ogni numero indica quanto è luminoso il pixel. Consideriamo ad esempio l'immagine larga 3 pixel e alta 2:

```
1 2 3
4 5 6
```

questo corrisponderebbe ad una variabile `struct image img`, con `img.rows = 2`, `img.cols = 3` e `img.data` che punta ad un area di memoria contenente i valori `{ 1, 2, 3, 4, 5, 6 }`.
La funzione accetta come parametro un puntatore ad una immagine e deve ritornare una immagine, allocata dinamicamente sull'heap, raddoppiandone la larghezza (colonne) e l'altezza (righe). Il raddoppio è equivalente a ricopiare prima le righe e poi le colonne o viceversa. Ad esempio l'immagine 2x2 seguente:

```
0 1
2 3
```

diventerebbe:

```
0 0 1 1
0 0 1 1
2 2 3 3
2 2 3 3
```

L'immagine in input sarà sempre un puntatore valido con dimensioni non nulle.

Esercizio 3 (7 punti)

Nel file `parole.c` implementare la definizione della seguente funzione:

```
extern void a_parole_1_999(char *sz, unsigned int i);
```

La funzione riceve in input il valore numerico `i` che deve essere compreso tra 1 e 999. In output produce nello spazio di memoria puntato da `sz` (che si assume essere già allocato e di dimensione sufficiente) la versione a parole in italiano in base dieci del numero `i`, come stringa C zero terminata.

Le regole per scrivere i numeri in italiano sono poche:

1) I numeri da 10 a 20 vanno scritti con i termini opportuni: dieci, undici, dodici, …, diciannove, venti.
2) Le unità da 1 a 9 vengono scritte con i termini opportuni: uno, due, …, nove.
3) Le decine vengono scritte con i termini opportuni (venti, trenta, …) a meno che non siano seguite da una unità 1 o 8, perché in quel caso perdono l'ultima lettera: trentuno invece che trentauno, settantotto invece che settantaotto.
4) Il numero 100 viene scritto come "cento", mentre le altre centinaia vengono scritte come le unità seguite dalla parola "cento": duecento, trecento, …, novecento.

Nei numeri non si devono inserire spazi. Se ad esempio `i=459`, `sz` deve essere riempito con "`quattrocentocinquantanove`". Ricordarsi il terminatore.
Se i non è tra 1 e 999 (inclusi), è necessario impostare `sz` alla stringa vuota. `sz` non sarà mai `NULL`.

Esercizio 4 (7 punti)

Nel file `commaseparatedvalues.c` implementare la definizione della seguente funzione:

```
extern double *read_csv(FILE *f, size_t *n);
```

La funzione riceve in input `f`, un puntatore a file aperto in lettura in modalità tradotta (testo), e `n`, un puntatore ad una variabile di tipo `size_t`. La funzione deve leggere da file una sequenza di valori numerici decimali separati da virgole, ritornare un puntatore ad un'area di memoria allocata dinamicamente su heap contenente i valori letti codificati come double e impostare la variabile puntata da n al numero di valori letti correttamente. I valori numerici utilizzano il punto decimale come indicatore della mantissa. Tra i numeri e le virgole di separazione possono essere presenti zero o più whitespace (spazio, tabulazione, newline, carriage return).

Ad esempio è valido il file contenente:

```
1,2,3,4,5
```

che produce il vettore di 5 double { 1.0, 2.0, 3.0, 4.0, 5.0 }. Ma è corretto anche il file:

```
3.21,4.3
,
9.01, 2.9 , -23 , 0
```

che produce il vettore di 6 double { 3.21, 4.3, 9.01, 2.9, -23.0, 0.0 }.

La funzione deve interrompersi appena trova qualcosa che non rispetta il formato fornito. Se nel file non c'è alcun valore valido, la funzione ritorna `NULL` e imposta il valore puntato da `n` a 0.

Esercizio 5 (7 punti)

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

```
extern void unpack_values(uint32_t pack, uint16_t *num1, uint16_t *num2, uint16_t *num3);
```

La funzione accetta in input un dato di tipo `uint32_t` contenente 3 numeri da 0 a 1023, così codificati:

numero del bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
significato	0	0	Numero 3										Numero 2										Numero 1									

In pratica 3 numeri non negativi a 10 bit vengono impacchettati in un solo valore a 32 bit. I bit 30 e 31 non vengono utilizzati e sono fissi a 0. La funzione deve estrarre i tre numeri e metterli nelle variabili a 16 bit puntate da `num1`, `num2` e `num3`.
Ad esempio se si codificano i numeri 1, 2 e 3 (in binario 0000000001, 0000000010, 0000000011) si ottiene il numero a 32 bit 00 0000000011 0000000010 0000000001 (gli spazi sono messi solo per far vedere dove finiscono i valori numerici), ovvero 0000.0000.0011.0000.0000.1000.0000.0001 (raggruppando per quattro), ovvero in esadecimale 0x00300801, in decimale 3147777. La funzione che riceve pack=3147777 deve ritornare in num1 il valore 1, in num2 il valore 2, in num3 il valore 3.