Esame di Laboratorio del 09/06/2021

Esercizio 1 (5 punti)

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

```
extern unsigned int conta_divisori(unsigned int num, unsigned int max);
```

La funzione `conta_divisori()` accetta come parametri due interi senza segno `num` e `max`, e restituisce il numero di divisori di `num` che non siano maggiori di `max`.

Ad esempio, invocando la funzione con:

```
num = 20
max = 100
```

Essa deve restituire:

```
6
```

in quanto i divisori di `num`, minori di `max`, sono

```
{ 1, 2, 4, 5, 10, 20}
```

Ovvero tutti i divisori di 20. Chiamandola invece con:

```
num = 20
max = 9
```

Essa deve restituire:

```
4
```

visto che solo i divisori:

```
{ 1, 2, 4, 5}
```

non sono maggiori di 9. 10 e 20 non vanno bene e non vengono conteggiati.

Esercizio 2 (6 punti)

Il Well-Known Text (WKT) è un linguaggio creato per rappresentare oggetti di geometria vettoriale (quali Punti, Linee e Poligoni) su una mappa.

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

```
struct point {
    double x, y;
};
```

e la funzione:

```
extern size_t write_point(FILE *f, const struct point *p, size_t n);
```

La `struct point` consente di rappresentare un punto nel piano cartesiano, caratterizzato dalle sue coordinate sull'asse x e sull'asse y.

La funzione `write_point()` accetta come parametri un puntatore a file `f` già aperto in scrittura in modalità tradotta (testo) ed un puntatore a vettore di `struct point` contenente n elementi.

La funzione scrive, per ognuno degli n elementi contenuti in p , una riga di testo contenente le due coordinate del punto in base 10 (x e successivamente y) separate da un singolo spazio, contenute in una parentesi tonda, e precedute dalla stringa `POINT`.

La funzione restituisce il numero di elementi correttamente salvati su file. Se, per qualsivoglia motivo, la funzione non scrive nessun elemento sul file, essa quindi restituisce `0`.

Ad esempio, dato in input il vettore di 3 punti:

$$\{(5,1),(4,3.7),(6.3,8)\}$$

la funzione scriverà su file le righe:

```
POINT(5 1)↵
POINT(4 3.7)↵
POINT(6.3 8)↵
```

e restituirà 3. Con il simbolo ↵ indichiamo il ritorno a capo. La rappresentazione dei numeri può essere ottenuta con il modificatore %g della funzione `fprintf()`.

Esercizio 3 (7 punti)

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

```
struct segmento {
    int a, b;
};
```

e la funzione:

```
extern struct segmento *intersezione(struct segmento s1, struct segmento s2);
```

La `struct segmento` consente di rappresentare un segmento, definito in una sola dimensione, compreso tra la coppia **ordinata** di punti `a` e `b`.

La funzione riceve in input due `struct segmento` `s1` e `s2`, e restituisce un puntatore a una `struct segmento` allocata dinamicamente che rappresenta l'intersezione tra `s1` e `s2`. Qualora non esista intersezione tra i due segmenti, la funzione restituisce `NULL`.

Un segmento rappresenta tutti i valori interi da `a` (incluso) a `b` (escluso), ovvero il segmento $[7, 10)$ è l'insieme di numeri `{7, 8, 9}`. L'intersezione di due segmenti è l'insieme dei valori contenuti in entrambi i segmenti e può sempre essere rappresentata con un segmento, se non è vuota.

Ad esempio dati i due segmenti $[1, 4)$ e $[3, 5)$, la funzione deve restituire il segmento $[3, 4)$, ovvero, l'intersezione tra gli insiemi `{1, 2, 3}` e `{3, 4}` è l'insieme che contiene solo il 3.

Dati invece $[1, 3)$ e $[5, 7)$ la funzione dovrà restituire `NULL`. Anche per i segmenti $[1, 5)$ e $[5, 7)$ non c'è intersezione.

Un segmento è valido solo se `a < b`. Qualora uno dei segmenti in input non sia valido, la funzione restituisce `NULL`.

Esercizio 4 (7 Punti)

Creare il file `hexstring.c` che consenta di utilizzare la seguente funzione:

```
extern void hexstring2vec(const char *s, uint8_t out[8]);
```

La funzione accetta come input una stringa C contenente da 0 a 8 coppie di caratteri che rappresentano cifre esadecimali, ovvero da `'0'` a `'9'` e da `'A'` a `'F'` e da `'a'` a `'f'`. In output deve riempire gli 8 byte puntati da `out` con i valori corrispondenti ad ogni coppia di caratteri in `s` interpretata come numero esadecimale a due cifre. Se `s` contiene meno di 8 coppie di cifre esadecimali, i restanti byte di `out` dovranno essere riempiti di zeri.

Alcuni esempi:

```
s                  | out
------------------ | -------------------------------------------------
"12AB34CD56EF7890" | { 0x12, 0xAB, 0x34, 0xCD, 0x56, 0xEF, 0x78, 0x90}
"12ab34cd56ef7890" | { 0x12, 0xAB, 0x34, 0xCD, 0x56, 0xEF, 0x78, 0x90}
"35AF"             | { 0x35, 0xAF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}
"0A0a0B0bcCdD"     | { 0x0A, 0x0A, 0x0B, 0x0B, 0xCC, 0xDD, 0x00, 0x00}
""                 | { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}
```

La stringa `s` sarà sempre formattata correttamente, ovvero il puntatore sarà valido, conterrà solo lettere da A a F minuscole o maiuscole o numeri, conterrà sempre un numero pari di cifre esadecimali e mai superiore a 16.

Esercizio 5 (8 Punti)

Un formato binario compatto per memorizzare valori reali nell'intervallo `[-2, 2)` con precisione circa alla quarta cifra decimale è ottenuto prendendo numeri **a 16 bit** in complemento a 2 e **dividendoli per** 2^{14}.

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

```
struct dvec {
    size_t n;
    double *d;
};
```

e la funzione:

```
struct dvec *read_dvec_comp(const char *filename);
```

La funzione apre il file `filename` in modalità non tradotta e legge dal file binario numeri a 16 bit in complemento a 2 in little endian, producendo in output una `struct dvec` allocata dinamicamente con `n` pari al numero di valori a 16 bit nel file, e `d` un puntatore ad un'area di memoria contenente gli `n` double corrispondenti ai valori codificati nel file.

I valori in `double` si ottengono prendendo i numeri a 16 bit con segno in complemento a 2 e dividendoli per 2^{14}.

Se non è possibile aprire il file, o il file non contiene alcun valore, la funzione crea una `struct dvec` con `n=0` e `d=NULL`. Ad esempio il file: `00 80 CD AB FF FF 00 00 01 00 32 54 FF 7F` contiene i valori (rappresentati in base 10 con 6 cifre decimali): `-2.000000, -1.315613, -0.000061, 0.000000, 0.000061, 1.315552, 1.999939`

Come è possibile vedere dal'esempio, non c'è alcun header, né alcuna informazione ulteriore nel file. Solo valori a 16 bit.