Fondamenti di Programmazione

A.A. 2018/2019

Appello del 19 Febbraio 2019 - Fila 1

NOME _____

COGNOME _____

MATRICOLA _____

EMAIL _____

Il presente documento deve essere consegnato al docente, mentre il codice sorgente associato alla prova d'esame deve essere memorizzato in un singolo file (es: *svolgimento.c*).

La consegna del codice sorgente avviene seguendo il collegamento presente sul desktop "Consegna Esami", inserendo le proprie credenziali e caricando il file (è necessario caricare solo ed esclusivamente il file sorgente) nella directory EsameFP190219_COGNOMEDOCENTE_Fila1

PROVA D'ESAME

Sia dato un insieme di punti in uno spazio 3D, ognuno dei quali identificato in modo non ambiguo dal nome di un colore. I punti sono memorizzati nel file `punti.txt`, dove ogni riga del file contiene le coordinate *x*, *y*, *z* (float) e il *nome* del colore del punto. Non è noto a priori il numero di punti memorizzati nel file. Ad esempio:

```
1.2   5.7   2     bianco
2.4   -5    1.5   nero
3     3.5   2.4   verde
-1.3  7.1   1.9   rosso
1.2   -3    3.4   giallo
-4.3  4.1   -2    marrone
```

Per semplicità, assumiamo che tutti i nomi di colori siano in lowercase, non contengano spazi e siano lunghi al massimo 20 caratteri, e che i dati di un punto siano separati da spazi o tabulazioni.

Scrivere il programma che, data una lettera dell'alfabeto inserita dall'utente, calcola la matrice delle distanze tra le coppie di punti il cui nome contiene tale lettera.

La matrice delle distanze `D` è una matrice quadrata `zxz`, dove `z` è la cardinalità dell'insieme di punti considerati, in cui l'elemento in posizione *(i,j)* rappresenta la distanza euclidea tra l'*i*-esimo e il *j*-esimo punto dell'insieme. La distanza euclidea tra due punti `a` e `b` è:

$$dist(a,b) = \sqrt{\sum_{i=1}^{3}(a_i - b_i)^2}$$

Nell'esempio precedente, assumendo che l'utente inserisca la lettera ***e***, l'output del programma consisterà nella matrice delle distanze tra i punti n***e***ro, v***e***rd***e***, marron***e*** (`z`=3, matrice 3x3) ossia:

```
0.000000; 8.568547; 11.830047
8.568547; 0.000000; 8.544589
11.830047; 8.544589; 0.000000
```

Scrivere il codice C che realizza il programma, basato sulle seguenti strutture dati e funzioni, rispettando gli argomenti descritti nel testo.

Si definisca il tipo di dato `punto` che consiste in una `struct` i cui campi rappresentano i dati di un singolo punto 3D: `x`, `y`, `z`, `nome` (scegliendo opportunamente i tipi di dato, se non precedentemente specificati).

Nota: tutti gli array coinvolti nella prova si intendono di dimensione massima `N=10` (dove `N` è definito a inizio file), se non diversamente specificato (per le matrici, la dimensione massima sarà `NxN`).

Esercizio 1 [punti 8]
`main()` *Invoca le funzioni degli esercizi seguenti (in ordine).*

Tutte le variabili passate come parametri alle funzioni devono essere dichiarate in `main`. Il file `punti.txt` viene aperto e chiuso in `main`. Subito dopo aver aperto il file, viene invocata la funzione dell'Esercizio 2 e viene stampato a video un messaggio di errore nel caso in cui tale funzione restituisca -1, oppure viene stampato il numero di punti correttamente caricati. Successivamente, viene chiesto all'utente di inserire una lettera dell'alfabeto, verificando che si tratti effettivamente di una lettera "minuscola" (e non di un numero o altro carattere). Se la verifica fallisce, viene chiesto nuovamente di inserire una lettera e si procede a nuova verifica. Infine, vengono invocate le funzioni degli Esercizi 3 e 4.

Esercizio 2 [punti 8]
`carica_dati(…)` *Carica i punti in un array di strutture.*

Legge il file `punti.txt`. In particolare, riceve come parametri il puntatore al file, un array di strutture di tipo `punto` e un intero `n` corrispondente al numero massimo di elementi memorizzabili in tale array. I dati di ogni punto vengono memorizzati in un elemento dell'array di strutture. Se il file non esiste, la funzione restituisce -1, altrimenti viene restituito il numero di punti letti con successo. Se una linea del file non rispetta il formato precedentemente descritto, la lettura si interrompe in quel punto (viene ritornato il numero di punti letti correttamente). Vengono considerati al massimo `n` punti (i successivi sono ignorati).

Esercizio 3 [punti 10]
`calcola_distanze(…)` *Calcola la matrice delle distanze tra alcune coppie di punti.*

Devono essere considerati solo i punti il cui nome contiene (almeno una volta) una certa lettera minuscola (aiuto: la funzione **strchr(stringa, carattere)**, in string.h, restituisce un valore diverso da NULL quando la "**stringa**" contiene il "**carattere**"). Riceve come parametri un array di strutture di tipo `punto` e il numero di punti memorizzati. Riceve anche una lettera (char) e una matrice quadrata di numeri reali (float) dove verranno memorizzate le distanze. Infine, troviamo un parametro di tipo intero che rappresenterà il numero di punti selezionati (parametro in uscita). La funzione è di tipo `void`.

Esercizio 4 [punti 6]
`stampa(…)` *Stampa la matrice delle distanze nel formato dell'esempio proposto.*

Riceve come parametri una matrice quadrata di distanze e la sua dimensione (`dimensione` = z^2, dove `z` è il numero di elementi coinvolti nel calcolo delle distanze). La matrice viene stampata a video separando gli elementi di una riga con il carattere *;* (punto e virgola) e uno spazio, analogamente a quanto riportato nell'esempio iniziale.