

Elaborazione Matrice di Reali (Matrice e Array)

Prova d'Esempio — main.c

Università di Catania — Dipartimento di Matematica e Informatica

Corso di Laurea in Informatica

Programmazione 1 e Laboratorio (Canale O-Z)

Descrizione del Programma

Si scriva un programma in linguaggio C, chiamato `main.c`, per l'analisi e l'elaborazione statistica di matrici numeriche contenenti valori reali a doppia precisione.

Il programma deve ricevere le dimensioni della matrice (n e m) e il nome del file contenente i dati tramite riga di comando, allocare dinamicamente una matrice bidimensionale, caricarvi i dati dal file, contare per ciascuna riga quanti valori ricadono in un intervallo proporzionale alla dispersione dei dati di quella riga, ordinare questi conteggi in ordine crescente e calcolarne la distribuzione cumulativa prima di salvare i risultati in un file di output.

Requisiti di Implementazione

Il programma deve essere strutturato nei seguenti punti:

PUNTO A: [3 punti] Gestione Parametri (CLI): Validare i parametri da riga di comando.

- Il programma deve ricevere esattamente tre argomenti: le dimensioni n e m della matrice (come interi) e il nome del file di input (es. `input.txt`).
- Controllare che n e m siano compresi tra 3 e 7 inclusi e che il nome del file abbia estensione `.txt`. In caso di errore (numero errato di parametri, valori fuori intervallo o estensione non valida), stampare un messaggio su `stderr` e terminare con exit code 1 (per parametri o valori errati) o 2 (per estensione errata).
- Stampare a schermo i valori dei parametri confermati.

PUNTO B: [8 punti] Struttura ADT Matrice Dinamica:

- **Punto 2a [4 punti]:** Allocare dinamicamente in memoria una matrice X di double di dimensione $n \times m$ tramite puntatori a puntatori (`double**`).
- **Punto 2b [4 punti]:** Aprire il file di input, leggerne le righe ed effettuarne il parsing popolandolo la matrice. Ciascuna riga del file contiene m valori reali separati da virgole. Se il file non può essere aperto, stampare un errore ed uscire con exit code 3. Stampare a schermo la matrice letta.

PUNTO C: [8 punti] Calcolo Statistico e Ordinamento:

- **Punto 3a [4 punti]:** Allocare dinamicamente un array Y di n interi. Inserire in ciascun elemento $Y[i]$ il numero di elementi della riga i -esima di X che hanno un valore compreso nell'intervallo aperto (a, b) , dove:

$$a = q - (\max - \min) \cdot 0.3, \quad b = q + (\max - \min) \cdot 0.3$$

con q , $\min$ e $\max$ che rappresentano rispettivamente la media, il minimo e il massimo valore della riga i -esima. Stampare a schermo l'array Y generato.

- **Punto 3b [4 punti]:** Ordinare l'array Y in-place in ordine crescente utilizzando l'algoritmo **Insertion Sort**. Stampare a schermo l'array Y ordinato.

PUNTO D: [8 punti] Somma Cumulativa ed Allocazione Array Z:

- **Punto 4a [4 punti]:** Allocare dinamicamente un array Z di n interi tramite l'uso della funzione `malloc`.
- **Punto 4b [4 punti]:** Popolare l'array Z calcolando la somma cumulativa degli elementi dell'array ordinato Y :

$$Z[i] = \sum_{j=0}^i Y[j]$$

Stampare a schermo l'array Z .

PUNTO E: [3 punti] Output su File: Salvare i risultati su file.

- Scrivere nel file `output.txt` il contenuto finale degli array Y (ordinato) e Z (cumulativo).
- Chiudere il file, liberare la memoria occupata dalla matrice X e dagli array, stampare un messaggio di conferma e terminare con exit code 0.

Specifiche delle Funzioni

Il programma dovrà essere articolato in un unico file sorgente e contenere almeno le seguenti funzioni con opportuni parametri formali:

- **decodeParameters**: riceve `argc` e `argv`, valida i parametri CLI e restituisce le variabili tramite puntatore o struttura dedicata.
- **readFile**: legge il file di input e restituisce la matrice X allocata dinamicamente.
- **getArray**: riceve la matrice X e restituisce l'array Y allocato dinamicamente con il conteggio degli elementi negli intervalli.
- **getCumulative**: riceve l'array Y e restituisce l'array Z contenente la somma cumulativa.
- **insertionSort**: ordina in-place l'array Y in ordine crescente.
- **writeOutput**: scrive gli array Y e Z sul file di output.

Note

- **Durata della prova**: 120 minuti.
- È **VIETATO** usare variabili globali.
- Si inseriscano i file sorgenti direttamente nella propria home directory.
- Accesso alla documentazione disponibile tramite il browser all'indirizzo: <https://devdocs.io/c/>.
- È possibile ritirarsi inserendo la scritta "RITIRATO" come prima riga di ogni file sorgente.

Output di Controllo (Esempio di Esecuzione)

Caso 1 — Esecuzione corretta con parametri validi ed elaborazione completa

Eseguendo il programma con il comando:

```
./main 7 7 input.txt
```

Un esempio di output stampato a schermo dal programma è il seguente:

```
=====Stampa Parametri=====  
n = 7
```

```

m = 7
filename = input.txt

=====Stampa X=====
0.24 -0.84 0.34 -0.51 0.67 0.04 -0.78
0.38 -0.95 0.45 0.31 -0.22 0.17 -0.74
0.22 0.08 -0.55 0.73 0.64 -0.10 0.52
-0.28 0.76 0.41 0.00 -0.20 0.75 -0.66
0.51 0.05 -0.86 0.60 -0.28 0.25 -0.48
-0.92 0.39 0.04 -0.43 0.21 -0.63 0.99
-0.74 -0.69 0.62 0.94 -0.18 0.35 -0.58

=====Stampa Y=====
3 3 4 4 3 4 2

=====Stampa Y ordinato=====
2 3 3 3 4 4 4

=====Stampa Z=====
2 5 8 11 15 19 23

```

Output salvato con successo in output.txt

In questo scenario, il file `output.txt` generato conterrà un testo simile al seguente:

```

Y: 2 3 3 3 4 4 4
Z: 2 5 8 11 15 19 23

```

(Exit code atteso: 0)

Caso 2 — Esecuzione con estensione parametri errata

Eseguendo il programma con il comando:

```
./main 7 7 input.pdf
```

Il programma stampa su `stderr`:

```
Errore: il file deve avere estensione .txt
```

(Exit code atteso: 2)

Appendici

Questo testo contiene le istruzioni e il codice di emergenza forniti per superare eventuali blocchi logici.

APPENDICE A: Dati di Test (Bypass Lettura da File)

Se non riuscite a completare la lettura da file (Punto 2b) o se riscontrate crash (Segmentation Fault) nel parsing, potete copiare ed incollare questa funzione nel vostro codice per popolare la matrice con dati statici e proseguire. L'uso di questa funzione azzerà il punteggio del Punto 2b.

```
void carica_dati_mock(double** X, int n, int m) {
    double test_data[7][7] = {
        {0.236,-0.845,0.342,-0.513,0.666,0.039,-0.784},
        {0.382,-0.953,0.453,0.312,-0.224,0.169,-0.738},
        {0.223,0.076,-0.545,0.735,0.638,-0.105,0.522},
        {-0.278,0.759,0.412,0.004,-0.204,0.747,-0.656},
        {0.514,0.053,-0.865,0.604,-0.283,0.247,-0.478},
        {-0.915,0.386,0.042,-0.432,0.211,-0.627,0.986},
        {-0.738,-0.693,0.620,0.938,-0.180,0.354,-0.584}
    };
    for (int i = 0; i < n && i < 7; i++) {
        for (int j = 0; j < m && j < 7; j++) {
            X[i][j] = test_data[i][j];
        }
    }
    printf("\n[!] Dati di emergenza caricati (Bypass Lettura File).\n");
}
```

APPENDICE B: Strutture Dati Alternative (Downgrade ad Array)

Se l'allocazione dinamica delle strutture dinamiche (Punto 2a) o la loro elaborazione causano crash bloccanti, potete decidere di aggirare l'ostacolo dichiarando nel main un array statico a due dimensioni per contenere i dati letti dal file o simulati:

```
double matrice_emergenza[100][100];
```

Sviluppate i punti successivi lavorando su questa struttura statica. In questo caso, i punti relativi alla struttura dinamica (Punti 2a, 2b) saranno valutati 0 punti, e il voto massimo dell'intera prova sarà bloccato a 18/30.