Programozás 1

Gyakorló feladatok II. ZH-ra

Alsó háromszög átlaga

Készítsen egy olyan C nyelvű programot, amely meghatározza egy 5x5 méretű, `float` típusú elemekből álló négyzetes mátrix alsó háromszögében található értékek átlagát. Az alsó háromszögbe azok az elemek tartoznak, amelyek a mátrix főátlóján és annak baloldalán helyezkednek el.

A programban implementálja egy `lower_triangle_average` nevű függvényt a következő szignatúrával:

```
void lower_triangle_average(int n, float matrix[n][n], float *avg);
```

Ez a függvény fogadja a mátrix méretét (`n`), a mátrixot (`matrix`), valamint egy `avg` mutatót, ahová az alsó háromszög átlagát kell menteni.

A program:

- Hozzon létre egy 5x5-es mátrixot, amelyet véletlenszerű számokkal tölt fel a [-2.0, 2.0] intervallumból.
- A véletlenszám-generátort inicializálja a `2025` értékkel.
- Számolja ki az alsó háromszög átlagát a `lower_triangle_average` függvénnyel.
- Írassa ki a kapott eredményt két tizedesjegyre formázva.

Példa egy ilyen mátrixra:

$$\begin{bmatrix} 0.5 & 1.2 & -1.0 & 0.3 & -0.6 \\ -0.4 & -1.3 & 0.7 & 1.0 & 0.2 \\ 1.1 & 0.0 & 1.5 & -0.8 & 1.8 \\ -1.7 & -0.9 & 0.2 & -1.0 & 0.4 \\ 0.6 & 1.9 & -1.2 & 0.5 & 1.0 \end{bmatrix}$$

Példa a futásra:

```
$ ./a.out
A számított alsó háromszög átlaga: 0.05
```

Mátrix főátló alatti elemeinek szórása

Készítsen egy C nyelvű programot, amely kiszámolja egy 4x4-es lebegőpontos számokat tartalmazó mátrix főátló alatti elemeinek szórását!

A főátló alatti elemek azok, amelyek a mátrixban a főátló alatt helyezkednek el (azaz ahol a sorindex nagyobb, mint az oszlopindex).

Az eljárás szignatúrája:

```
void lower_triangle_stddev(int n, double matrix[n][n], double *stddev);
```

Az eljárás a `stddev` változóba mentse a kiszámított szórást!

A program:

- Hozzon létre egy 4x4-es mátrixot, amelyet véletlenszerű számokkal tölt fel a [1.0, 5.0] intervallumból.
- A véletlenszám-generátort inicializálja a `2025` értékkel.
- Számolja ki a főátló alatti elemek szórását.
- Írja ki az eredményt két tizedesjeggyel.

Példa egy ilyen mátrixra:

$$\begin{bmatrix} 4.1 & 3.2 & 2.9 & 1.5 \\ 2.8 & 3.7 & 4.0 & 2.1 \\ 3.3 & 2.6 & 4.4 & 3.9 \\ 1.9 & 2.5 & 3.1 & 4.2 \end{bmatrix}$$

A főátló alatti elemek szórása a következő képlettel számolható:

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2}$$

ahol:

- s: a szórás
- n: az elemek száma
- x_i: az i-edik elem
- $\bar{x}$: az elemek átlaga

Példa futási kimenet:

```
$ ./a.out
A főátló alatti elemek szórása: 0.57
```

Véletlenszerű szócsere

Készítsen egy C nyelvű programot, amely két parancssori argumentumot vár:

- Bemeneti fájl neve – ez a fájl tartalmazza az eredeti szöveget.
- Kimeneti fájl neve – ebbe a fájlba kerül a módosított szöveg.

A program működése:

- A program beolvassa a bemeneti fájl tartalmát.
- A szövegben található szavakat véletlenszerűen lecseréli egy előre megadott szavak listájából (pl. {“alma”, “körte”, “banán”, “szilva”}), előfordulhat, hogy a szövegben lévő szó nem cserélődik le.
- A cserék véletlenszerűek, tehát minden futtatás más eredményt ad.
- A módosított szöveget a megadott kimeneti fájlba menti.

Hibaellenőrzés:

- Ha a felhasználó nem ad meg fájlnevet, vagy a bemeneti fájl nem létezik, írjon ki hibaüzenetet a standard hibakimenetre.

Példa bemeneti fájl (`input.txt`):

```
Ez egy teszt szöveg, amit módosítani szeretnénk.
```

Példa futtatás:

```
$ ./a.out
Hiba! Adja meg a bemeneti fájlt és a kimeneti fájl nevét!

$ ./a.out example.txt out.txt
Hiba! A(z) 'example.txt' nevű fájl nem létezik vagy nem lehet megnyitható!

$ ./a.out input.txt output.txt
```

Példa kimeneti fájl (`output.txt`):

```
körte egy alma banán, amit szilva szeretnénk.
```

Véletlenszerű szöveg-tömörítés

Írjon egy olyan programot, amely két parancssori argumentumot vár (a sorrend rögzített):

- az első egy meglévő szöveges fájl neve, amelyet szeretnénk "tömöríteni",
- a második az a fájlnév, ahová a módosított tartalmat elmentjük.

A program feladata, hogy beolvassa a megadott forrásfájlt, majd az abban található minden szóból véletlenszerűen eltávolítson néhány karaktert, 30% eséllyel minden egyes karakterre. A program ezután a módosított szöveget a második argumentumban megadott fájlba írja.

Példák:

Bemeneti fájl (`input.txt`):

```
This is a sample text file to demonstrate random character removal.
Each character has a 30 percent chance to be deleted independently.
The output will vary each time you run the program.
```

Lehetséges kimenet (`output.txt`):

```
Thi  a smple txe fie to dmontrate ranom carater rmoal.
Eac charactr ha  30 pernt chnce to be dleted indenently.
Th outut wll var each tiem yo un the proram.
```

A kimenet minden futtatásnál eltérhet, mivel a törlés véletlenszerű.

Elvárások:

- Ha a felhasználó nem ad meg elég argumentumot, vagy a megadott forrásfájl nem létezik, írjon a standard hibakimenetre hibaüzenetet.
- A fájlkezelés során minden fájlt megfelelően zárjon le!
- A program minden futtatáskor más eredményt adjon (használjon véletlenszám-generátort).

Hibaüzenetek:

```
$ ./a.out
Hiba! Nem adtál meg fájlnevet.

$ ./a.out nemletezofajl.txt output.txt
Hiba! Ilyen fájl nem létezik.
```

Mozifilmek

Tekintsük a `mozik.csv` szöveges állományt. A fájl egyes sorai a következő adatokat tartalmazzák: film IMDb értékelése, film címe, szavazatok száma. A sorokban lévő adatok pontosvesszővel vannak egymástól elválasztva.

Írjon programot, ami kiírja a képernyőre azon filmek címeit, melyekre legalább fél millióan szavaztak!

Informatív módon azt is írja ki, hogy hány ilyen film található a `mozik.csv` állományban!

A filmek címei lexikografikusan csökkenő sorrendben legyenek kiírva.

Az egyes filmek adatai egy `Film` nevű struktúrában legyenek eltárolva.

A fájl beolvasása során dinamikus memóriafoglalással legyenek létrehozva a `Film` struktúrák.

Napi hőmérsékleti adatok

Egy meteorológiai rendszerhez készítsen C nyelvű programot, amely egy CSV formátumú fájlból (`temperatures.csv`) olvassa be több nap hőmérsékleti adatait.

A fájl minden sora tartalmaz egy dátumot és három hőmérsékleti értéket (reggel, délben, este), pontosvesszővel elválasztva, az első sor a fejléc.

A program töltse be az adatokat egy megfelelő struktúrába, számolja ki minden nap mérési értékeinek átlagát, majd rendezze a napokat az átlag alapján csökkenő sorrendbe.

A rendezett adatokat jelenítse meg, minden naphoz írja ki a dátumot, a három mérési értéket és az átlagot, jól elkülönítve egymástól.

Ha a fájl nem nyitható meg, jelenítsen meg hibaüzenetet, és fejezze be a futást szabályosan.

Példa bemeneti fájl (`temperatures.csv`):

```
Date;Morning;Noon;Evening
2022-01-01;5.3;12.0;8.6
2022-01-02;6.2;14.4;10.5
2022-01-03;7.8;15.1;11.3
2022-01-04;8.1;16.8;12.0
2022-01-05;9.4;18.3;13.4
```

Futási példa:

```
$ ./a.out
Dátum: 2022-01-05
Reggeli: 9.4 °C, Déli: 18.3 °C, Esti: 13.4 °C
Átlagos hőmérséklet: 13.70 °C
---
Dátum: 2022-01-04
Reggeli: 8.1 °C, Déli: 16.8 °C, Esti: 12.0 °C
Átlagos hőmérséklet: 12.30 °C
---
Dátum: 2022-01-03
Reggeli: 7.8 °C, Déli: 15.1 °C, Esti: 11.3 °C
Átlagos hőmérséklet: 11.40 °C
---
Dátum: 2022-01-02
Reggeli: 6.2 °C, Déli: 14.4 °C, Esti: 10.5 °C
Átlagos hőmérséklet: 10.37 °C
---
Dátum: 2022-01-01
Reggeli: 5.3 °C, Déli: 12.0 °C, Esti: 8.6 °C
Átlagos hőmérséklet: 8.63 °C
```

Napi lépésszám-elemzés

Készítsen C nyelvű programot, amely egy `daily_steps.csv` nevű fájlból több hét napi lépésszám-adatait dolgozza fel. A fájlban minden sor pontosvesszővel elválasztva tartalmazza a hét sorszámát, a hét napjának nevét és az adott nap megtett lépésszámát egész számként. Az első sor a fejléc.

Olvassa be a fájl minden sorát egy struktúrákat tartalmazó tömbbe, ahol minden rekord a hét sorszámát, a nap nevét és a lépésszámot tárolja.

Minden héthez számolja ki a heti teljes és az átlagos lépésszámot.

Minden hétnél válassza ki azokat a napokat, ahol a napi lépésszám kevesebb, mint az adott heti átlag.

A kiválasztott napokat rendezze csökkenő sorrendbe a lépésszám értékei alapján.

Írja ki hétenként a teljes és az átlagos lépésszámot, majd az átlag alatt lévő napok listáját, ahol minden nap neve mellett a lépésszám szerepel.

Ha a program nem tudja megnyitni a CSV fájlt, jelenítsen meg hibaüzenetet, és lépjen ki szabályosan.

Példa bemeneti fájl (`daily_steps.csv`):

```
Week;Day;Steps
1;Monday;8321
1;Tuesday;7642
1;Wednesday;10543
1;Thursday;6211
1;Friday;8890
1;Saturday;4780
1;Sunday;9350
2;Monday;8450
2;Tuesday;7021
2;Wednesday;11000
2;Thursday;6300
2;Friday;9000
2;Saturday;5000
2;Sunday;9600
```

Példa futási kimenet:

```
$ ./a.out
--- Week 1 ---
Heti teljes lépésszám: 55937 lépés
Heti átlagos lépésszám: 7991 lépés
A következő napokon volt az átlag alatt a lépésszám:
Tuesday:  7642 lépés
Thursday: 6211 lépés
Saturday: 4780 lépés

--- Week 2 ---
Heti teljes lépésszám: 55371 lépés
Heti átlagos lépésszám: 7910 lépés
A következő napokon volt az átlag alatt a lépésszám:
Tuesday:  7021 lépés
Thursday: 6300 lépés
Saturday: 5000 lépés
```

Szöveg tokenizálása

Készítsen egy programot, amely egy bemeneti fájlból beolvas egy szöveget, majd a szöveget szavakra (tokenekre) bontja, és minden szót új sorba ír egy kimeneti fájlba.

A program a következő két parancssori argumentumot várja:

- az első a bemeneti fájl neve (pl. `input.txt`)
- a második a kimeneti fájl neve (pl. `output.txt`)

A tokenizálás során a szavakat a következő elválasztó karakterek mentén bontsa fel:

- szóköz (`' '`),
- tabulátor (`'\t'`),
- újsor (`'\n'`),
- írásjelek: `',' '.' ';' ':' '!' '?'`

Példa

Bemenet (`input.txt`):

```
Ez egy gyakorló feladat, amely szöveg feldolgozást tanít!
```

Kimenet (`output.txt`):

```
Ez
egy
gyakorló
feladat
amely
szöveg
feldolgozást
tanít
```

Ha a felhasználó nem ad meg elég argumentumot, vagy a bemeneti fájl nem létezik, a program írjon egy megfelelő hibaüzenetet a standard hibakimenetre.

A kimeneti fájlba csak nem üres tokeneket írjon.

Minden szó külön sorban szerepeljen.

Használjon `fgets()` és `strtok()` függvényeket a megvalósításhoz.

Hibaüzenetek:

```
$ ./a.out
Hiba! Nem adtál meg fájlnevet.

$ ./a.out nincs.txt output.txt
Hiba! Ilyen fájl nem létezik.
```

Egyszerű műveletek

Hozzon létre egy `math_utils.h` nevű header fájlt, amely tartalmazza az alábbi függvények deklarációját:

- `int add(int a, int b);`
  Két egész szám összegét adja vissza.
- `int subtract(int a, int b);`
  Két egész szám különbségét adja vissza.
- `int multiply(int a, int b);`
  Két egész szám szorzatát adja vissza.
- `float divide(int a, int b);`
  Két pozitív egész szám hányadosát adja vissza lebegőpontos értékként. Ha a második szám `0`, akkor a függvény térjen vissza egy speciális értékkel (például `-1.0f`), vagy jelezze a hibát más módon.

Gondoskodjon arról, hogy a header fájlban legyenek `include guardok`, hogy elkerüld a többszörös beillesztést.

Hozzon létre egy `math_utils.c` nevű forrásfájlt, amely megvalósítja a `math_utils.h`-ban deklarált függvényeket.

Írjon egy `main.c` nevű fájlt, amely:

- Beimportálja a `math_utils.h` header fájlt.
- Meghívja mind a négy függvényt különböző bemenetekkel (például: 10 és 5), és kiírja a kapott eredményeket a képernyőre.

Elvárások:

- A header fájl legyen jól strukturált, legyenek benne `include guardok`.
- A függvények megfelelően legyenek deklarálva és definiálva.
- A `main.c` fájlban jól látható legyen a függvényhívás és az eredmény kiírása.
- A program forduljon le hiba nélkül, és fusson helyesen.

Példa kimenet:

```
$ gcc main.c math_utils.c -o main
$ ./main
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 2.00
```