Felvételi, 2025. július
Szoftverfejlesztés mesterképzés
Javítókulcs

I. Programozás és programozási nyelvek

```c
void osszefesul_kiir(int tomb1[], int meret1, int tomb2[], int meret2) {
    int i = 0, j = 0;

    while (i < meret1 || j < meret2) {
        int ertek;
        int db = 0;

        if (i < meret1 && (j >= meret2 || tomb1[i] < tomb2[j])) {
            ertek = tomb1[i];
        } else if (j < meret2 && (i >= meret1 || tomb2[j] < tomb1[i])) {
            ertek = tomb2[j];
        } else {

            ertek = tomb1[i];
        }

        while (i < meret1 && tomb1[i] == ertek) {
            db++;
            i++;
        }
        while (j < meret2 && tomb2[j] == ertek) {
            db++;
            j++;
        }

        printf("%d (%d) ", ertek, db);
    }

    printf("\n");
}
```

Javítókulcs / Értékelési szempontok:

	Pont
Függvény aláírása helyes (void típus, megfelelő paraméterek)	1 pont
Ciklus helyesen halad végig a két tömbön egyszerre, ellenőrzi ha még van	3 pont
Helyes logika az aktuális minimum kiválasztásához	2 pont

Helyes előfordulásszámlálás (ugyanazon érték ismétlései a két tömbben)	2 pont
Kiírás formátuma megfelelő (szám + zárójelben előfordulás)	1 pont
Végeredmény pontosan megfelel a példa szerintinek	1 pont
Összesen	**10pont**

II. Adatbázisok

Adott a **Párizsi Olimpiai Játékok** részleges adatbázisa:

Venues(venue, disciplines, date_start, date_end) **Medal_info**(code, name)

Athletes(code, name, gender, function, country_code, nationality_code, height, weight, disciplines, events, birth_date, occupation, lang) **Gender**(id, name)

Results(rdate, event_name, discipline_name, venue, participant_code, participant_type, rank, result, result_type)

Coaches(code, name, gender, function, country_code, discipline, birth_date)

Medals(medal_code, medal_date, discipline, event, winner_code)

Teams(code, team, country_code, discipline, events, athletes_code)

Tokyo_medals(country_code, gold_medal, silver_medal, bronze_medal)

Countries(code, country, country_long, population, gdp, latitude, longitude)

Írj SQL utasításokat a következő feladatok megoldására:

a) Tudva azt, hogy az **Athletes** tábla lang oszlopa tartalmazza a sportolók által beszélt nyelveket vesszővel elválasztva, írasd ki annak a sportolónak a nevét, aki a legtöbb nyelven beszél. Név és nyelvek száma formában. **(1 pont)**

```
select name, length(NVL(lang, '')) - length(replace(NVL(lang, ''), ',', '')) + 1 as
LanguageCount

from   athletes

order by LanguageCount desc nulls last

fetch first row only;
```

b) Írd ki azoknak a romániai női sportolóknak a nevét, akik csapatsportban vettek részt és érmet szereztek. **(1 pont)**

```
select a.name
from olimpia.athletes a join olimpia.teams t
on a.acode = t.athletes_code
join olimpia.medals m on t.code =m.winner_code
where a.gender = 1 and a.country_code like 'ROU';
```

c) A sportolók hány százaléka nő és hány férfi? **(1 pont)**

```
with a as (select count(*) as male_count
        from olimpia.athletes where gender =0),
        b as (select count(*) as female_count
        from olimpia.athletes where gender =1),
        g as (select count(*) as all_count
        from olimpia.athletes)
select round(a.male_count*100/g.all_count,2) as "Ferfiak szazaleka" ,
round(b.female_count*100/g.all_count,2) as "Nok szazaleka"
from a,b,g;
```

d) Melyik az az ország, amely 1 érmet sem szerzett? **(1 pont)**

```
with individual_winners as (select a.country_code as country
                    from athletes a
                     join medals m on a.acode = m.winner_code),
        team_winners as (select distinct country_code as country
                from teams t
                join medals m on t.code = m.winner_code)
select code from olimpia.countries
minus
```

```
        (select country from individual_winners
        union
        select country from team_winners);
```

e) Sorold fel azt az öt országot, ahol a legnagyobb javulás az aranyérmek számában a tokiói olimpiához képest! **(2 pont)**

```
with
    a as (select t.country_code as country_code,count(distinct m.winner_code) as csapat
        from teams t join medals m on t.code = m.winner_code
        where m.medal_code=1
        group by t.country_code ),
    b as (select t.country_code as country_code,count(*) as egyeni
        from athletes t join medals m on t.acode = m.winner_code
        where m.medal_code=1
        group by t.country_code),
    c as (select country_code,gold_medal from tokyo_medals )
select a.country_code, abs(a.csapat+b.egyeni-c.gold_medal)  as kulonbseg
from a join c on a.country_code=c.country_code
join b on a.country_code=b.country_code
order by kulonbseg desc
fetch first 5 row only;
```

f) Írassuk ki azokat a sportolókat, akik több mint 5 érmet szereztek. **(2 pont)**

```
        select name, count(*)
        from (select a.name, m.medal_code
            from athletes a
                join teams t on (a.acode = t.athletes_code)
                join medals m on (t.code = m.winner_code)
            union all
```

```
    select a.name, m.medal_code
    from athletes a
        join medals m on (a.acode = m.winner_code))
group by name
having count(*) >= 5
order by 2 desc, 1;
```

g) Mennyi a korkülönbség a legfiatalabb és legidősebb olimpikon között. **(1 pont)**

```
select round((max(birth_date)-(min(birth_date)))/365) as korkülönbség
from athletes;
```

III. Objektumelvű programozás és szoftvertervezés

1. Az **egységbezárás** azt jelenti, hogy egy osztály belső állapotát (változóit) elrejtjük a külvilág elől, és csak **publikus metódusokon keresztül** (getter/setter) tesszük elérhetővé vagy módosíthatóvá. Ez segíti az adatok védelmét, a hibák megelőzését és a karbantarthatóságot.

Rossz példa (nem tartja be az elvet):

```
public class Person {
    public String name;
    public int age;
}
```

Ebben a példában az `age` és `name` mezők nyilvánosak (`public`), így közvetlenül elérhetők és módosíthatók kívülről, ami megsérti az egységbezárás elvét.

2.

```
import java.util.Arrays;

class Util {

   public static double median(double[] numbers) {
        if (numbers == null || numbers.length == 0) {
            throw new IllegalArgumentException("A bemeneti tömb nem lehet üres.");
        }
        double[] sorted = Arrays.copyOf(numbers, numbers.length);
        Arrays.sort(sorted);
        int n = sorted.length;
        if (n % 2 == 1) {
            return sorted[n / 2];
        } else {
            return (sorted[n / 2 - 1] + sorted[n / 2]) / 2.0;
        }
    }
}

public class Main{
   public static void main(String args[]) {
       double[] array = {30, 20, 10};
       try{
           System.out.println(Util.median(array));
       } catch(IllegalArgumentException e) {
           e.printStackTrace();
       }
   }

}
```

3.

```
public class Singleton {
    // Privát, statikus példány, kezdetben null
    private static Singleton instance;

    // Privát konstruktor - így kívülről nem lehet példányosítani
    private Singleton() {
        System.out.println("Singleton példány létrehozva.");
    }

    // Nyilvános, statikus metódus a példány lekérésére
    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();  // Csak egyszer hozza létre
        }
        return instance;
    }

}
```

Magyarázat:

- **Privát konstruktor:**
  Nem lehet kívülről példányosítani az osztályt (`new Singleton()` nem engedélyezett más osztályból).
- **Statikus példány (`instance`):**
  Az osztály saját magának tárolja az egyetlen példányát.
- **`Statikus getInstance()` metódus:**
  Ez a metódus ellenőrzi, hogy létezik-e már példány: Ha **nem**, akkor létrehozza. Ha **igen**, akkor a meglévőt adja vissza.
- **Garancia az egy példányra:**
  A felhasználó mindig ugyanazt a példányt kapja vissza, így az erőforrásokat meg lehet osztani, például konfiguráció vagy naplózó (`Logger`) esetén.

4.

A **Dependency Inversion Principle** a SOLID alapelvek egyik tagja, és azt mondja ki, hogy:

> **A magas szintű modulok ne függjenek alacsony szintű moduloktól. Mindkettő függjön absztrakciótól (pl. interfésztől). Az absztrakciók nem függhetnek a részletektől. A részletek függjenek az absztrakcióktól.**

Ez segít a **rugalmas, jól bővíthető és tesztelhető** kód kialakításában.

```
// Alacsony szintű modul (konkrét adatbázis-kezelés)
public class MySQLDatabase {
    public void saveData(String data) {
        System.out.println("Adat elmentve MySQL-be: " + data);
    }
}
```

```java
// Magas szintű modul
public class DataService {
    private MySQLDatabase db = new MySQLDatabase();

    public void store(String data) {
        db.saveData(data);
    }
}
```

A `DataService` (magas szintű modul) **közvetlenül** függ a `MySQLDatabase` osztálytól (alacsony szintű modul).
Ez **merev kapcsolat**: ha más adatbázist akarunk használni (pl. `PostgreSQLDatabase`), módosítani kell a `DataService` kódját.
A rendszer **nehezen tesztelhető** vagy bővíthető.

5.

Túlterhelés:

- **Egy osztályon belül** ugyanaz a metódusnév, de **különböző paraméterlisták**.
- **Fordítási időben** (compile time) dől el, melyik metódus hívódik meg.

Felülírás:

- Egy **származtatott osztály** újradefiniálja az ősosztály metódusát **ugyanazzal a szignatúrával**.
- **Futásidőben** (runtime) dől el, melyik metódus hívódik.

```java
class Animal {
    public void makeSound() {
        System.out.println("Az állat hangot ad.");
    }
}

class Dog extends Animal {
    // Felülírás (overriding)
    @Override
    public void makeSound() {
        System.out.println("A kutya ugat.");
    }

    // Túlterhelés (overloading)
    public void makeSound(String mood) {
        System.out.println("A kutya " + mood + " hangon ugat.");
    }
}
```

Tulajdonság	Overriding (Felülírás)	Overloading (Túlterhelés)
Hol alkalmazzuk?	Öröklésnél (alaposztály → leszármazott)	Egy osztályon belül
Metódusnév	Ugyanaz	Ugyanaz
Paraméterlista	Ugyanaz	Különböző
Visszatérési típus	Lehet azonos vagy kompatibilis	Nem elég megkülönböztetésre
Mikor dől el?	Futásidőben (runtime)	Fordítási időben (compile time)