BABEŞ-BOLYAI TUDOMÁNYEGYETEM
MATEMATIKA ÉS INFORMATIKA KAR

Licenszvizsga 2026
Informatika - magyar nyelv

1. VÁLTOZAT

MEGJEGYZÉSEK

- Mindegyik tétel kötelező; a tételeknél teljes megoldásokat kérünk.
- A dolgozat minimális átmenő jegye az ötös (5.00).
- Munkaidő: három (3) óra.

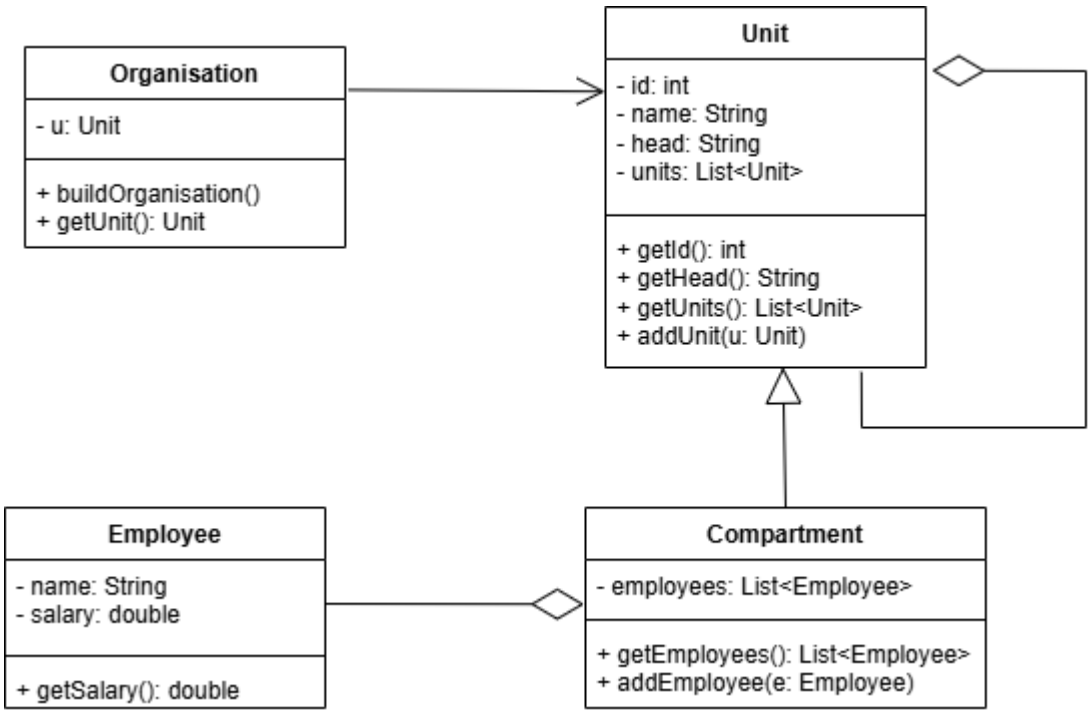
Algoritmusok és programozás TÉTEL

- **A használt programozási nyelvet meg kell jelölni.**
- **A megírandó programokban nem szükséges a dinamikusan lefoglalt memória felszabadítása.**
- **A megfelelő programozási stílus hiánya (azonosítók beszédes elnevezése, indentálás, megjegyzések ha szükségesek, a kód olvashatósága) az adott tételhez tartozó pontszám 10%-nak az elvesztéséhez vezet.**
- **Tilos új adattagok és metódusok hozzáadása a kijelentésben említetteken kívül, leszámítva a konstruktorokat. Tilos a kijelentésben megadott adattagok és metódusok láthatóságának a megváltoztatása.**
- **Előre definiált rendezett konténerek és rendezési valamint keresési műveletek használata tilos.**
- **Adattípusokhoz használhatóak a megfelelő programozási nyelvek (Python, C++, Java, C#) létező könyvtárai.**

1. **(3 pont)** Írjatok függvényt a **Python, C++, Java** vagy **C#** programozási nyelvek valamelyikében, amely paraméterként megkap egy csökkenő sorrendbe rendezett, valós számokból álló *v* sorozatot (amely egy részleg alkalmazottainak fizetéseit tartalmazza) és egy *salariu* valós számot (amely egy új alkalmazott fizetését tartalmazza). A függvény be kell szúrja a *salariu* értéket a *v* sorozatba. A beszúrás olyan módon kell történjen, hogy a *v* sorozat csökkenő sorrendben rendezett maradjon az alkalmazottak fizetése szerint. Használjatok egy iteratívan implementált $O(\log_2 n)$ időbonyolultságú segédfüggvényt, amely visszatéríti azt a pozíciót, ahová a *salariu* értéket be kell szúrni a *v* sorozatba, ahol *n* a *v* sorozat hossza. *A rekurzív megoldások részpontszámot érnek.*
   Megjegyzés. Egy elem beszúrása a *v* sorozat *k*-adik pozíciójára a *k*, *k*+1, ..., *n* pozíciókon lévő elemek eggyel jobbra való eltolását feltételezi, ami után a sorozat hossza 1-gyel nő. ***Tilos előre definiált beszúró függvények használata egy elemnek a sorozat belsejébe történő beszúrására.***

2. **(1 pont)** Adjátok meg az 1-es pontbeli függvény időbonyolultságát a *legjobb*, *átlag* és *legrosszabb* esetben. Indokoljátok a választ.
3. **(5 pont)** Adott az alábbi UML diagram, mely tartalmazza az **Organisation** (Szervezet, egy szervezet hierarchikus felépítését ábrázolja), **Unit** (Szervezeti egység), **Compartment** (Részleg) és **Employee** (Alkalmazott) osztályokat.

Az osztályok konstruktorai nincsenek feltüntetve a diagramon.



- Az **Organisation** osztály egy szervezetet ábrázol (egy szervezeti egységekből álló hierarchikus struktúra formájában). A **getUnit()** metódus visszatéríti az osztály *u* adattagját.
  - A **buildOrganisation()** metódus létrehozza a szervezet hierarchikus felépítését. *Ezt a metódust nem kell implementálni.*
- A **Unit** osztály egy szervezeti egység hierarchikus felépítését ábrázolja. Ez rendelkezik egy azonosítóval (*id*), egy névvel (*name*) és egy igazgatóval (akinek a neve a *head* adattagban van eltárolva). Az egység közvetlen alárendeltségébe több (al)egység tartozik (melyek a *units* adattagban vannak eltárolva). Az osztály tartalmaz metódusokat az *id*, *head* és *units* adattagok elérésére, valamint egy **addUnit**(*u*: **Unit**) metódust, amely hozzáadja az *u* egységet a *units* lista végéhez.
- A **Compartment** osztály egy olyan egységet ábrázol, amelynek már nincsenek újabb (al)egységek az alárendeltségében. Ez az *employees* adattagban alkalmazottak (**Employee**) egy listáját tárolja el. Az *employees* alkalmazotti lista a fizetéseik csökkenő sorrendjében van eltárolva. Az osztály rendelkezik egy **addEmployee**(*e*: **Employee**) metódussal, amely hozzáad egy alkalmazottat a rendezett *employees* listához, valamint egy **getEmployees**() metódussal, amely visszatéríti a részleg alkalmazottainak listáját.
- Az **Employee** osztály egy alkalmazottat ábrázol, akit egy név (*name*) és egy fizetés (*salary*) jellemez. Az osztály rendelkezik egy **getSalary**() metódussal, amely visszatéríti a *salary* adattagot.

Megjegyzés: *A szervezeten* (**Organisation**) *belüli egységek* (**Unit**) *különböző azonosítókkal (id) rendelkeznek.*

Írjatok programot a **C++**, **Java** vagy **C#** programozási nyelvek valamelyikében, az alábbi követelmények szerint:

a) Deklaráljátok az összes osztályt, adattagot és metódust a fenti diagramnak megfelelően. Csak a **Compartment** osztály konstruktorát kell implementálni.

b) Definiáljatok egy függvényt, amely paraméterként egy **Organisation** típusú *o* objektumot és egy *n* egész számot kap, és visszatérít egy listát, amely tartalmazza az összes olyan alkalmazottat, aki (közvetlenül vagy közvetve) az *n* azonosítójú (*id*) szervezeti egység (Unit) alárendeltségébe tartozik. A visszatérített lista üres lesz abban az esetben, ha a szervezetnek nincs *n* azonosítójú egysége.

c) Definiáljatok egy függvényt, amely paraméterként egy **Organisation** típusú *o* objektumot kap, és visszatéríti az *o* szervezetben lévő részlegek igazgatóinak listáját.

d) Hozzatok létre egy **Organisation** típusú *o* objektumot, hívjátok meg a **buildOrganisation**() metódust, majd hívjátok meg a b) és c) pontokban definiált függvényeket a létrehozott szervezetre.

Adatbázisok TÉTEL **1. VÁLTOZAT**

1. Feladat (4 pont)

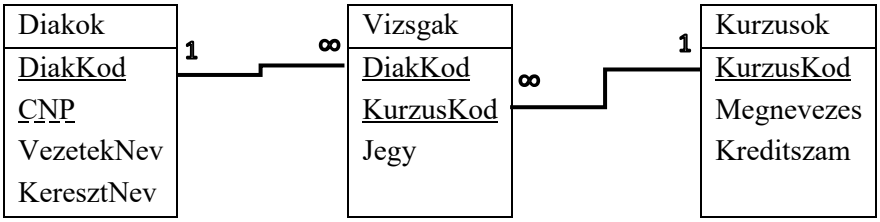
Egy kutatóintézet nyilvántartja a saját kutatási tevékenységei során használt növénygyűjteményeivel kapcsolatos információkat. A növénygyűjtemények az intézet épületének különböző tárolóhelyiségeiben vannak tárolva, és ezeket a kutatóintézet alkalmazottai használják. Az intézet alkalmazottaival, tárolóhelyiségeivel, növénygyűjteményeivel, növényekkel és földrajzi régiókkal kapcsolatos információkat egy relációs adatbázisban tárolják:

- A kutatóintézet alkalmazottai esetén érdekel: alkalmazott kódja, neve, e-mail címe (mely minden alkalmazott esetén egyedi), születési dátuma, a kutatóintézetben betöltött pozíciója és fizetése.
- Minden tárolóhelyiség esetén tároljuk: kódját, megnevezését, leírását, valamint a kezeléséért felelős alkalmazottat. Egy alkalmazott több tárolóhelyiség kezeléséért is felelhet, de egy tárolóhelyiséget csak egy alkalmazott kezelhet.
- Minden földrajzi régiónak van: kódja, megnevezése, leírása, valamint országa.
- Minden növénygyűjtemény esetén érdekel: kódja, címe, leírása, valamint a kutatóintézet gondozásába vételének dátuma. Egy adott növénygyűjtemény az intézet egy adott tárolóhelyiségében kerül elhelyezésre. Minden tárolóhelyiség több növénygyűjteménynek is helyet adhat, de egy növénygyűjtemény csak egyetlen tárolóhelyiségben tárolható.
- Egy növényről érdekel: növény kódja, tudományos elnevezése, közismert elnevezése, osztálya, rendje, valamint az az érték, amely jelzi, hogy a növény védett-e vagy sem. Egy növénygyűjtemény több növényt is tartalmazhat, és egy növény is szerepelhet több növénygyűjteményben. Egy adott növény egy adott növénygyűjteményhez legfeljebb egyszer rendelhető hozzá. Minden növény- növénygyűjtemény párra vonatkozóan (pl. az n1 növény és ngy1 növénygyűjtemény alkotta páros esetén) tárolni szeretnénk a növénynek a növénygyűjteménybe való kerülésének dátumát, valamint a földrajzi régiót, ahonnan az adott növényt gyűjtötték.

Tervezzük meg a fenti információknak megfelelő relációs adatbázis sémáját, melynek táblái BCNF-ben vannak. Jelöljük az elsődleges- és külső kulcsokat, valamint a relációk további kulcsjelöltjeit. A relációk sémáját az alábbiakban megadott módok egyikének megfelelően adjuk meg:

* példa a Diakok, Kurzusok és Vizsgak táblákra vonatkozóan:

V1. Adatbázis-diagram a táblák megadásával: az elsődleges kulcsok egyenes vonallal, míg a tábla további kulcsjelöltjei szaggatott vonallal vannak aláhúzva; a fiú táblában megadott külső kulcsot az apa táblában szereplő elsődleges kulccsal/kulcsjelölttel egy direkt él köti össze (lsd. a Vizsgak tábla DiakKod attribútuma és a Diakok tábla DiakKod attribútuma közötti él).



V2.

Diakok[DiakKod, CNP, Vezeteknev, Keresztnev]

Kurzusok[KurzusKod, Megnevezes, Kreditszam]

Vizsgak[DiakKod, KurzusKod, Jegy]

Az elsődleges kulcsok alá vannak húzva egyenes vonallal, míg a további kulcsjelöltek szaggatott vonallal. {DiakKod} külső kulcs a Vizsgak relációban, mellyel a Diakok tábla {DiakKod} attribútumára hivatkozunk. {KurzusKod} külső kulcs a Vizsgak relációban, mellyel a Kurzusok tábla {KurzusKod} attribútumára hivatkozunk.

2. Feladat (5 pont)

Tekintsük az alábbi relációsémákat, melyek egy múzeum műalkotásairól és időszakos kiállításairól tárolnak információkat:

Kiallitasok[KiallitasKod, Cim, Leiras, MegnyitoDatuma, TeremSzam]
MuveszetiIranyzatok[IranyzatKod, Megnevezes]
Muveszek[MuveszKod, Nev, SzuletesiDatum, *IranyzatKod*]
Mualkotasok[MualkotasKod, Cim, AlkotasEv, *MuveszKod, KiallitasKod*, MualkotasTipus]

Az elsődleges kulcsok alá vannak húzva, míg a külső kulcsok dőlt betűvel vannak szedve, nevük pedig megegyezik azon attribútumok nevével, amelyekre hivatkoznak.

a. Írjunk egy SQL lekérdezést, mely megadja azon kiállításokat (kiállítás címe, megnyitó dátuma, terem száma), ahol látható legalább egy, „Pablo Picasso" nevű művésztől származó műalkotás, de ahol egyetlen, „Claude Monet" nevű művésztől származó műalkotás sincs kiállítva! A lekérdezés szűrje ki a duplikátumokat!

b. Tekintsük a MuveszetiIranyzatok, Muveszek és Mualkotasok relációk alábbi előfordulásait:

MuveszetiIranyzatok:

IranyzatKod	Megnevezes
1	Kubizmus
2	Impresszionizmus
3	Posztimpresszionizmus

Muveszek:

Muvesz Kod	Nev	Szuletesi Datum	Iranyzat Kod
1	Pablo Picasso	1881.10.25	1
2	Claude Monet	1840.11.14	2
3	Vincent van Gogh	1853.03.30	3
4	Edgar Degas	1834.07.19	2

Mualkotasok:

Mualkotas Kod	Cim	Alkotas Ev	Muvesz Kod	Kiallitas Kod	Mualkotas Tipus
1	Guernica	1937	1	1	Festmeny
2	Jacqueline with Flowers	1954	1	1	Festmeny
3	Water lilies	1919	2	2	Festmeny
4	The Little Fourteen-Year-Old Dancer	1880	4	2	Szobor
5	The Ballet Class	1874	4	2	Festmeny
6	Sunflowers	1888	3	3	Festmeny
7	The Starry Night	1889	3	2	Festmeny
8	Cafe Terrace at Night	1888	3	1	Festmeny

b1. A fentebb megadott reláció előfordulások esetén adjuk meg az alábbi lekérdezés kiértékelésének eredményét! Adjuk meg az oszlop(ok) nevét és értékét! A választ NEM kell megindokolni!

```
SELECT M.Nev, MI.Megnevezes, COUNT(MA.MualkotasKod) AS Szam
FROM Muveszek M
        INNER JOIN MuveszetiIranyzatok MI ON M.IranyzatKod = MI.IranyzatKod
        INNER JOIN Mualkotasok MA ON MA.MuveszKod = M.MuveszKod
GROUP BY M.MuveszKod, M.Nev, MI.Megnevezes
HAVING COUNT(MA.MualkotasKod) >=
 (SELECT AVG(T.Szam)
  FROM      (SELECT M1.MuveszKod, COUNT(*) AS Szam
             FROM Muveszek M1
                    INNER JOIN MuveszetiIranyzatok MI1 ON M1.IranyzatKod = MI1.IranyzatKod
                    INNER JOIN Mualkotasok MA1 ON MA1.MuveszKod = M1.MuveszKod
             GROUP BY M1.MuveszKod
                    ) T
    )
```

b2. Döntsük el, hogy a Mualkotasok reláció fentebb megadott előfordulása kielégíti-e az alábbi funkcionális függőségeket vagy sem! Indokoljuk meg a választ!

- { MualkotasKod } → { MuveszKod }
- { KiallitasKod } → { MualkotasKod }.

Operációs rendszerek TÉTEL

1. feladat (5 pont) Válaszoljunk az `./a.out` program végrehajtásával kapcsolatos alábbi kérdésekre, tudva, hogy az az itt megadott forráskódból lett lefordítva. Feltételezzük, hogy minden szükséges fejlécállomány jelen van, továbbá a `fork` és `write` rendszerhívások sikeresen hajtódnak végre.

```
 1  int main() {
 2    int i;
 3    char buf[3] = {'L','0','\n'};
 4    for (i = 0; i < 3; i++) {
 5      buf[1] = '0' + i;
 6      write(i%2+1, buf, 3);
 7      fork();
 8    }
 9    write(1, "END\n", 4);
10    for (i = 0; i < 3; i++)
11      wait(NULL);
12    return 0;
13  }
```

a) Hányszor lesz a kimenetre írva az `END` sor az `./a.out` végrehajtásakor? Indokoljuk a választ.

b) Hányszor jelenik meg az `out` állomány tartalmában külön-külön az `L0`, `L1` és `L2` sor, az `./a.out > out` végrehatása után? Indokoljuk a választ.

c) Az eredeti folyamatot nem számítva, hány folyamat jön létre a végrehajtás során? Indokoljuk a választ.

d) Magyarázzuk el részletesen a `10-11` sorok ciklusának működését és határozzuk meg, hogy hány közvetlen gyerekfolyamatra vár az eredeti folyamat. Indokoljuk a választ.

e) Összesen hányszor lesznek az `./a.out` végrehajtása sorá kiírva az `L`-lel kezdődő sorok`(L0+L1+L2)`, amennyiben a `6.` sort áthelyezzük közvetlenül a `7.` után? Indokoljuk a választ.

2. feladat (4 pont) Válaszoljunk az alábbi `./a.sh` nevű UNIX Shell szkript végrehajtásával kapcsolatos kérdésekre, feltéve, hogy a megadott `f.txt` állományt használja. Megjegyzés: a sorszámok NEM tartoznak az állományhoz.

```
1  #!/bin/bash
2  while read NAME GRADE; do
3    if [ $GRADE -ge 5 ]; then
4      echo "$NAME"
5    fi
6  done < f.txt | sort | uniq -c | sort -nr | head -n 1
```

`f.txt`

```
1  Anca 7
2  Radu 9
3  Anca 8
4  Radu 2
5  Anca 3
```

a) Mi lesz a kimenete csak a `while` ciklusnak (a `2-6` sorokban az első `| sort` előtti részt tekintjük)? Indokoljuk a választ.
b) Mi a `sort | uniq -c` kifejezés hatása? Indokoljuk a választ.
c) Mit ír ki a teljes szkript? Indokoljuk a választ.
d) Hogyan módosul az eredmény, ha eltávolítjuk a `uniq -c` előtti `sort` parancsot? Indokoljuk a választ.

BAREM INFORMATICĂ
VARIANTA 1

Subiect Algoritmică și Programare

Oficiu – **1p**

Cerința **1. – 3p**

Funcția de bază – **0.4p** din care
- signatura – 0.1p
- implementare – 0.3p

Funcția auxiliară – **2.6p** din care
- signatura – 0.1p
- implementare iterativă funcție auxiliară având complexitate timp $O(\log_2 n)$ – 2.5p
  * soluție recursivă având complexitate timp $O(\log_2 n)$ – 1p
  * soluție iterativă având complexitate timp $O(n)$ – 0.5p

Cerința **2. – 1p**
- caz favorabil **0.3p** din care
  - complexitate – 0.15
  - justificare – 0.15
- caz mediu **0.4p** din care
  - complexitate – 0.2
  - justificare – 0.2
- caz defavorabil **0.3p** din care
  - complexitate – 0.15
  - justificare – 0.15

Cerința **3.a) – 1.7p**

Definirea clasei Organisation – **0.2p** din care
- atribut – 0.1
- metode **buildOrganisation**, **getUnit** – 0.1

Definirea clasei Unit – **0.5p** din care
- atribute – 0.2
- constructor – 0.1
- metode **getId, getHead, getUnits, addUnit** – 0.2

Definirea clasei Compartment– **0.7p** din care
- relația de moștenire – 0.2
- atribut – 0.1
- **constructor (a1)** – 0.3
- metode **getEmployees, addEmployee** – 0.1

Definirea clasei Employee – **0.3p** din care
- atribute– 0.2
- metoda **getSalary** – 0.1

Funcția **3.b)** – **2p**
- signatura – **0.1p**
- implementare funcție – **1.8p**
- returnare rezultat – **0.1p**

Funcția **3.c)** – **1p**
- signatura – **0.1p**
- implementare funcție – **0.8p**
- returnare rezultat – **0.1p**

Funcția principală **3.d)** – **0.3p**
- construire obiect **o** – 0.1p
- apel metoda **buildOrganisation**() – 0.1p
- apel funcții b) și c) – 0.1p

BAREM INFORMATICĂ
VARIANTA 1

Subiect Baze de date
Oficiu – **1p**

Problema 1. Punctaj - 4p
- relații cu atribute corecte, chei primare, chei candidat: **3p**
- legături modelate corect (chei externe): **1p**

Problema 2. Punctaj - 5p
- **a** - rezolvarea completă a interogării: **2.5p**

- **b1** - rezultat evaluare interogare:

Nume	**Denumire**	**Nr**
Pablo Picasso	Cubism	2
Vincent van Gogh	Postimpresionism	3
Edgar Degas	Impresionism	2

  - coloane – **0.5p**
  - valori tuplu – **1p**

- **b2 -** { CodOperaDeArta } → { CodArtist } este satisfăcută – **0.25p**; **0.25p** explicație
  - { CodExpozitie } → { CodOperaDeArta } nu este satisfăcută – **0.25p**; **0.25p** explicație

Notă: La specializările Informatică engleză și Informatică maghiară se iau în considerare versiunile traduse în limbile corespunzătoare.

BAREM INFORMATICA
VARIANTA 1

Sisteme de Operare

1p – oficiu

Problema 1 (5p)

1p – a) 8 = 2^3 procese

1p – b) L0 o dată, L1 niciodată, L2 de patru ori

1p – c) 7 procese fii, 2^3 fără procesul inițial

1p – d) se așteaptă 3 procese fiu, dacă acestea nu există, wait returnează -1; procesul inițial așteaptă 3 procese copil

1p – e) Liniile L se afișează în total de 14 ori (L0×2 + L1×4 + L2×8)

Problema 2 (4p)

1p – a) Anca, Radu, Anca (în această ordine, câte una pe linie); numele doar pentru liniile cu nota >=5

1p – b) `sort` sortează în odine alfabetică liniile primite, iar `uniq` elimină liniile consecutive identice și afișează de câte ori apare fiecare, folosind opțiunea `-c`

[opțional: Aici obținem: 2 Anca şi 1 Radu (numărul precedă numele);]

1p – c) 2 Anca - `sort -nr` ordonează descrescător numeric după numărul din față, `head -n 1` păstrează prima linie

1p – d) Neavând linii consecutive identice, uniq -c produce 1 Anca, 1 Radu, 1 Anca, rezultatul final fiind 1 Anca