BABEŞ-BOLYAI TUDOMÁNYEGYETEM
MATEMATIKA ÉS INFORMATIKA KAR

Licenszvizsga
Informatika - magyar nyelv

2. VÁLTOZAT

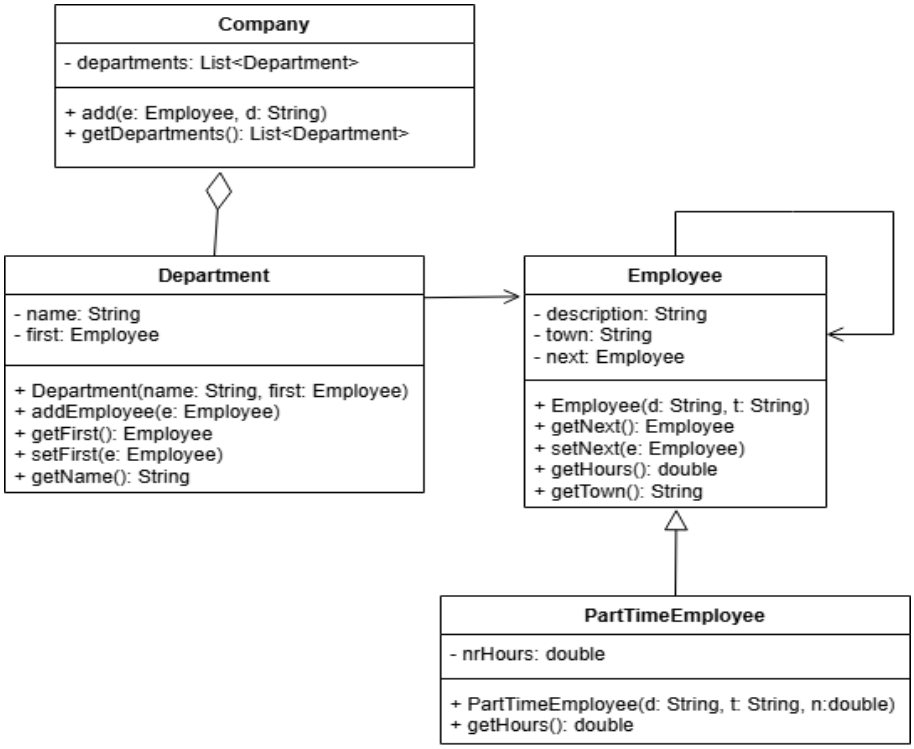
MEGJEGYZÉSEK

- 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. (**2.5 pont**) Adjatok meg egy függvényt a **Python, C++, Java** vagy **C#** programozási nyelvek valamelyikében, amely paraméterként megkapja a *lst* (nem feltétlenül láncolt) listát, amely karakterláncokat tartalmaz abécé sorrendbe rendezve (melyek egy cég részlegeinek nevét jelölik) és egy *nev* karakterláncot, amely egy részleg nevét jelöli. Feltételezve (a függvény előfeltételeként), hogy a *nev* karakterlánc nem szerepel a *lst* listában, a függvény be kell szúrja az *lst* listába a *nev* karakterláncot. A beszúrás olyan módon kell történjen, hogy az *lst* lista rendezett maradjon a részlegek neve szerinti ábécésorrendben. Használjunk rekurzív implementálási módot. Segédfüggvények implementálása megengedett. Az iteratív megoldások részpontszámot érnek.
2. (**1 pont**) Adjátok meg az **1**-es pontnál megírt függvény időbonyolultságát a *legjobb*, *átlag* és *legrosszabb* esetben. Indokoljátok a választ.
3. (**5.5 pont**) Adott az alábbi UML diagram, mely tartalmazza a **Company** (vállalat), **Department** (Részleg – az alkalmazottak egy csoportja), **Employee** (FőállásúAlkalmazott) és **PartTimeEmployee** (RészmunkaidősAlkamlazott) osztályokat.



- Az **Employee** osztály *description* (munkakör leírása) és *town* (származási város), valamint a **Department** osztály *name* (részleg neve) adattagjai nem lehetnek üresek, míg a **PartTimeEmployee** osztály *nrHours* adattagja (az alkalmazott havi munkaóráinak száma) egy szigorúan pozitív, 160-nál kisebb szám kell legyen. A megfelelő osztályok konstruktorai érvényesítik a fenti feltételeket.
- Az **Employee** osztály tartalmaz egy **getHours()** metódust, amely 160-at térít vissza, valamint egy **getTown()** metódust, amely visszatéríti az alkalmazott származási városát. A **getNext()**, illetve **setNext(e: Employee)** metódusok visszatérítik, illetve beállítják a *next* adattagot.
- A **PartTimeEmployee** osztály **getHours()** metódusa visszatéríti a részmunkaidős alkalmazott havi munkaóráit (*nrHours* adattag).
- A **Department** osztály a részleg (teljes vagy részmunkaidős) alkalmazottai nyilvántarását tárolja. Az **addEmployee(e: Employee)** metódus hozzáad egy alkalmazottat a részleghez, míg a **getName()** metódus visszaadja a részleg nevét. A **getFirst()**, illetve **setFirst(e: Employee)** metódusok visszatérítik, illetve beállítják a *first* adattagot.
- A **Company** osztály a vállalaton belüli részlegek listáját tárolja el. Az **add(e: Employee, d: String)** metódus hozzáadja az *e* alkalmazottat a *d* nevű részleghez (ha nem létezik ilyen nevű részleg, akkor az létrejön). A **getDepartments()** metódus visszatéríti a vállalaton belüli részlegek listáját.

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 következő metódusokat kell implementálni:
a1) Az **Employee** osztály **setNext(e: Employee)** metódusát.
a2) A **Department** és **PartTimeEmployee** osztályok konstruktorait.
a3) A **Department** osztály **addEmployee(e: Employee)** és **setFirst(e: Employee)** metódusait. Az **addEmployee(e: Employee)** metódus esetén a megengedett időbonyolultság a legrosszabb esetben $\theta(1)$. Azon megoldások, amelyek nem teljesítik a fenti korlátot csak részpontot érnek.
a4) A **Company** osztály **add(e: Employee, d: String)** metódusát.

b) Definiáljatok egy függvényt, amely paraméterként egy **Company** objektumot kap, és megszámolja azt a maximális számú alkalmazottat (m), akik ugyanabból a városból származnak, és kiírja az összes olyan várost, ahonnan pontosan m alkalmazott származik (egy város csak egyszer jelenjen meg). Feltételezzük, hogy egy város neve legtöbb 20 karakter hosszúságú, a fenti metódus esetén a megengedett időbonyolultság átlagos esetben $\theta(n)$ (ahol n a vállalat alkalmazottainak száma). Azon megoldások, amelyek nem teljesítik a fenti korlátot csak részpontot érnek.

c) Hozzatok létre egy **Company** típusú objektumot, amelyhez adjatok hozzá 4 alkalmazottat, amelyből 2 legyen részmunkaidős alkalmazott (tetszés szerint). Ezt követően hívjátok meg a **b)** pontban definiált függvényt.

Adatbázisok TÉTEL **2. VÁLTOZAT**

1. Feladat (4 pont)

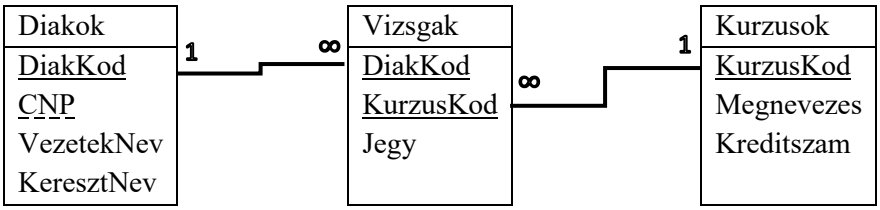
Egy szoftvercég különböző iparágak számára fejleszt alkalmazásokat, az adott iparág igényeihez igazított technológiák alkalmazásával. Az egyes részlegeken dolgozó alkalmazottaknak a cég projektjeihez való hozzárendelésének koordinálására a vállalat egy relációs adatbázist használ:

- Az alkalmazottak esetén érdekel: alkalmazott kódja, felvételi dátuma, neve (egyszerű attribútum), e-mail címe (egyedi minden alkalmazott esetén), valamint havi fizetésének értéke. Egy alkalmazott egy adott részlegen dolgozik és egy adott projekthez van hozzárendelve.
- Minden részlegnek van kódja és megnevezése. Egy adott részleg egyszerre több projekttel is foglalkozik.
- Egy projekt esetén tároljuk: projekt kódját, megnevezését (egyedi a vállalaton belül), költségvetését, a domináns technológiát (pl. P1T1 nevű projekt esetén a domináns technológia a ".NET"). Egy projekttel a cégen belül egy adott részleg foglalkozik.
- A vállalat több telephellyel is rendelkezik, ahol a cég alkalmazottai dolgozhatnak. Minden telephelyhez tartozik egy kód, egy cím (egyszerű attribútum) és egy telefonszám (egyedi minden telephely esetén). Egy adott telephely esetén érdekel, hogy rendelkezik-e vagy sem parkolóval, illetve üzemel-e benne kávézó vagy sem.
- Egy adott telephelyen több részleg alkalmazottai is dolgozhatnak, de minden telephely esetén a kezdetektől fogva rögzítve lett, hogy mely részlegek tevékenykedhetnek az adott telephelyen. Egy adott telephelyhez több részleg is hozzárendelhető, míg egy adott részleghez is rendelhető több telephely.

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:

Szerzok[SzerzoKod, Nev, SzarmazasiOrszag, SzuletesiEv]

Konyvek[KonyKod, Konyvcim, Szakterulet, Oldalszam, *SzerzoKod*]

Kiadok[KiadoKod, KiadoNev]

KonyvKiadasok[*KonyvKod*, *KiadoKod*, Ar, MegjelenesiEv]

Az elsődleges kulcsok alá vannak húzva, míg a külső kulcsok dőlt betűvel vannak szedve, nevük pedig megegyezik a hivatkozott tábla elsődleges kulcsának nevével.

a. Írjunk egy SQL lekérdezést, amely megadja azon szerzők kódját és nevét, akik esetén létezik legalább 1 olyan év, amikor legalább 2 könyvet írtak, valamint akiknek legalább egy könyvük a "KN2" nevű kiadónál jelent meg!

b. Tekintsük a Konyvek, Kiadok, KonyvKiadasok relációk alábbi előfordulásait:

Konyvek

Konyv-Kod	Konyv-cim	Szak-terulet	Oldal-szam	Szerzo-Kod
1	C1	SzT2	125	Sz1
2	C2	SzT1	175	Sz1
3	C3	SzT2	210	Sz3
4	C4	SzT3	98	Sz2
5	C5	SzT2	113	Sz4

Kiadok

KiadoKod	KiadoNev
K1	KN1
K2	KN2
K3	KN3

KonyvKiadasok

KonyvKod	KiadoKod	Ar	MegjelenesiEv
1	K2	34	2021
2	K1	28	2022
3	K3	55	2021
4	K2	21	2024
5	K1	42	2023

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 t1.SzerzoKod, t1.Konyvcim, t1.KonyvKod
FROM Konyvek t1
WHERE
EXISTS
    (SELECT KonyvKod FROM KonyvKiadasok t2
     INNER JOIN
     (SELECT kk.KiadoKod FROM KonyvKiadasok kk
          GROUP BY kk.KiadoKod
          HAVING COUNT(*) > 1 AND SUM(kk.Ar) < 60
       UNION ALL
       SELECT ki.KiadoKod FROM Kiadok ki WHERE ki.KiadoNev = 'KN2'
       ) t3 ON t3.KiadoKod = t2.KiadoKod
            WHERE t2.KonyvKod = t1.KonyvKod
    )
```

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

- {Oldalszam} → {KonyvKod}
- {SzerzoKod} → {Szakterulet}

2. VÁLTOZAT

Operációs rendszerek TÉTEL

1. Feladat (5 pont) Válaszoljunk az alábbi, eredetileg paraméterek nélkül meghívott `./a` programmal kapcsolatos kérdésekre, feltéve, hogy minden szükséges fejlécállomány jelen van, továbbá a `fork` rendszerhívás és legalább egy `exec` rendszerhívás sikerrel hajtódik végre.

```
 1  int main(int argc, char** argv) {
 2      if(argc > 1) {
 3          printf("%s\n", argv[1]);
 4      }
 5      fork();
 6      execlp("a", "a", "y", NULL);
 7      execl("./a", "./a", "z", NULL);
 8      printf("%d\n", wait(NULL));
 9      return 0;
10  }
```

a) Magyarázzuk el részletesen a 6. sor működését.

b) Hányszor lesz a standard kimenetre írva az `argv[1]` értéke? Indokoljuk a választ.

c) Mi lesz a standard kimenetre írva, ha a `PATH` környezeti változó nem tartalmazza az aktuális katalógust?

d) Hány folyamat fogja végrehajtani a `wait` rendszerhívást?

e) Mi lesz a standard kimenetre írva, ha a `PATH` környezeti változó tartalmazza az aktuális katalógust, és az 5. sort töröljük?

2. Feladat (4 pont) Válaszoljunk az alábbi UNIX Shell szkript végrehajtásával kapcsolatos kérdésekre, feltéve, hogy az `a.txt` állomány létezik, és minden sora szóközökkel elválasztott betűsorokat tartalmaz.

```
 1  #!/bin/bash
 2
 3  while read L; do
 4    R=""
 5    for W in $L; do
 6      X=""
 7      C=`echo $W | sed -E "s/(.)/\1 /g"`
 8      for A in $C; do
 9        X="$A$X"
10      done
11      R="$X $R"
12    done
13    echo $R
14  done < a.txt
```

a) Milyen értékeket fog felvenni az L és a W változó?

b) Magyarázzuk el részletesen a 7. sor működését.

c) Mi lesz a standard kimenetre írva a szkript végrehajtása nyomán, ha az `a.txt` tartalma az alábbi?

```
abc def
pqr xyz
```

d) Mit lesz a standard kimenetre írva a szkript végrehajtása nyomán, a c) pontban megadott `a.txt` bemeneti állományra, ha a 7. sorban töröljük az `1` számjegy utáni szóközt?

BAREM INFORMATICĂ
VARIANTA 2

Subiect Algoritmică și Programare

Oficiu – **1p**

Cerința **1.** **– 2.5p**

- signatura – **0.2p**
- implementare recursivă – **2.3p** din care
  - adăugare lista vidă – 0.5
  - adăugare la începutul/sfârșitul listei (pentru caz favorabil) – 0.5
  - apelul recursiv – 1.3

* implementare iterativă corectă – **0.7**p

Cerința **2.** **– 1p**

- caz favorabil **0.2p** din care
  - complexitate $\theta(1)$ – 0.1
  - justificare – 0.1
- caz mediu **0.4p** din care
  - complexitate $\theta(n)$ – 0.2 (n – dimensiunea listei *lst*)
  - justificare – 0.2
- caz defavorabil **0.4p** din care
  - complexitate $\theta(n)$ – 0.2 (n – dimensiunea listei *lst*)
  - justificare – 0.2

Cerința **3.a)** **– 2.6p**

Definirea clasei Employee – **0.6p** din care
- atribute – 0.1
- contructor – 0.1
- metoda **setNext (a1)** – 0.2
- metode **getNext**, **getHours**, **getTown** – 0.2

Definirea clasei PartTimeEmployee– **0.55p** din care
- relația de moștenire – 0.2
- atribut – 0.05
- **constructor (a2)** – 0.2
- metoda **getHours** – 0.1

Definirea clasei Department– **1p** din care
- atribute – 0.1
- **constructor (a2)**– 0.1
- metoda **addEmployee (a3)** – 0.5
  - * punctajul se acordă pentru implementare având complexitate $\theta(1)$
  - * pentru implementari având complexitate mai mare decat $\theta(1)$ se acorda 0.25p
- metoda **setFirst (a3) –** 0.2
- metode **getFirst**, **getName –** 0.1

Definirea clasei Company – **0.45p** din care
- atribut, metoda **getDepartments** – 0.1p
- metoda **add (a4)** – 0.35p

Funcția **3.b)** – **2.6p**

- signatura – **0.1p**
- implementare având complexitatea timp în caz mediu $\theta(n)$ – **2.5p**
  - * implementare având complexitatea timp în caz mediu $\theta(n\log_2 n)$ – **1.5p**
  - * implementare având complexitatea timp în caz mediu $\theta(n^2)$ – **0.75p**

Funcția principală **3.c)** – **0.3p**

- construire obiect de tip Company – 0.1p
- adăugare angajați - 0.1p
- apel funcție b) pentru expresie– 0.1p

BAREM INFORMATICĂ
VARIANTA 2

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:

CodAutor	Titlu	CodCarte
A1	T1	1
A2	T4	4

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

- **b2** - {NrPagini}→{CodCarte} este satisfăcută – **0.25p**; **0.25p** explicație
  - {CodAutor}→{Domeniu} 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.

VARIANTA 2

SUBIECT Sisteme de operare

Barem:

1p – oficiu

Problema 1 (5p)

1p – a) Execută programul `a` cu argumentul `y`, dacă locaţia programului `a` se găseşte in `PATH`

1p – b) Se va afişa de o infinitate de ori, pentru că programul se reexecută la linia 6 sau 7.

1p – c) Se vor afişa o infinitate de caractere `z`

1p – d) Nicunul pentru că unul dintre cele două apeluri sistem `execl` şi `execlp` se vor executa cu succes şi vor suprascrie codul procesului înainte să se ajungă la linia 8.

1p – e) Se vor afişa o infinitate de caractere `y`

Problema 2 (4p)

1p – a) L va lua ca valori fiecare linie din fişierul a.txt, variabila W va lua ca valori fiecare secvenţe de litere din linia stocată în L

1p – b) După fiecare literă din variabila W se adaugă un spaţiu, rezultatul stocându-se in variabila C

1p – c) Se va afişa

```
fed cba
zyx rqp
```

1p – d) Se va afişa

```
def abc
xyz pqr
```