Sapientia Erdélyi Magyar Tudományegyetem
Marosvásárhelyi Kar

Felvételi, 2025. július
– Alapképzés, informatika vizsga –

Hivatalból 1 pont jár. Munkaidő 3 óra.
Megjegyzések:

- A pszeudokód algoritmusban
  - Az „=” értékadást, az „==” pedig egyenlőség vizsgálatot jelent
  - A / operátorral osztási hányadost, a % operátorral pedig osztási maradékot kapunk meg
  - A „`minden i = 1, n végezd`” jelentése: minden $i = 1, 2, 3, \ldots, n$ értékre végezd el…
- **A programokat megjegyzésekkel kell ellátni!**
- **Törekedjünk hatékony megoldásokra!** (maximális pontszám feltétele)

1. (1 pont) Legyen az alábbi pszeudokód programrészlet:

```
res = 1
┌amíg n ≠ 0 végezd
| res = res * (n % 2)
| n = n / 10
└■
kiír res
```

Mit ír ki a fenti programrészlet, ha $n = 20240729$, illetve ha $n = 1315179$? Magyarázd egy mondatban, hogy mit határoz meg a fenti algoritmus.

0
1
A program meghatározza, hogy az adott szám összes számjegye *páratlan-e*; ha igen, akkor `1`-et, ha nem, akkor `0`-át ír ki.

2. (1 pont) Legyen az alábbi pszeudokód programrészlet:

```
res = 0
┌amíg x ≠ 0 végezd
| ┌ha (x % 10) > y && (x % 10) < z akkor
| | res = res * 10 + (x % 10)
| └■
| x = x / 10
└■
kiír res
```

Mit ír ki a fenti programrészlet, ha $x = 468297$, $y = 3$ és $z = 8$? Magyarázd egy mondatban, hogy mit határoz meg a fenti algoritmus.

`764`
A program kiválasztja azokat a számjegyeket az `x` számból, amelyek nagyobbak `y`-nál és kisebbek `z`-nél, majd ezekből a számjegyekből egy új számot állít össze az eredeti számjegyek sorrendjének fordítottjában.

3. (1 pont) Legyen az alábbi pszeudokód programrészlet, ahol a fibo(x) függvény igazat ad vissza, ha x a fibonacci sorozat egyik tagja (a fibonacci sorozat tagjai 21-ig: 0, 1, 1, 2, 3, 5, 8, 13, 21), ellenkező esetben hamisat:

```
count = 0
res = ...
┌minden i = 1, n végezd
| ┌ha fibo(a[i]) akkor
| | count = count + 1
| | ┌ ha a[i] > res akkor
| | | res = a[i]
| | └■
| └■
|
```

```
└■
kiír count, ' ', res
```

a) Mi kellene kerüljön a pontok helyére, ha azt szeretnénk, hogy a *res* változó az algoritmus végrehajtása után a legnagyobb, *a* tömbben szereplő fibonacci értéket tárolja.

b) Mit ír ki a fenti programrészlet (az *a)* pont szerinti értékadás mellette), ha az *a*[1..*n*] tömb tartalma:

i. 15, **3**, **2**, 88, 16, **34**, **1** ebben a sorrendben, és $n = 7$?

ii. 18, 32, 22, 11, 7 ebben a sorrendben, és $n = 5$?

```
res = -1
4 34
0 -1
```

4. (1 pont) Legyen az alábbi pszeudokód programrészlet:

```
┌minden i = 2, n végezd
|     ┌minden j = 2, n végezd
|     |     ┌  ha m[i-1][j] < m[i][j-1] akkor
|     |     |     m[i][j] = m[i-1][j]
|     |     |  különben
|     |     |     m[i][j] = m[i][j-1]
|     |     └■
|     └■
└■
```

Mi lesz az *m*[1..*n*][1..*n*] kétdimenziós tömb tartalma, ha kezdetben az első sora az 1, 3, 7, 2, 4 és első oszlopa pedig az 1, 8, 3, 2, 6 értékeket tartalmazza, ebben a sorrendben, és $n = 5$?

```
1   3   7   2   4
8   3   3   2   2
3   3   3   2   2
2   2   2   2   2
6   2   2   2   2
```

5. (1 pont) Írj Pascal vagy C/C++ **programot**, amely beolvas a billentyűzetről egy 1–12 közötti hónapszámot, és kiírja, hogy az adott hónap melyik negyedévhez tartozik (1, 2, 3 vagy 4).
Példa:

Bemenet:

```
7
```

Kimenet:

```
3
```

```
int honap;
cin >> honap;
cout << (honap - 1) / 3 + 1;
```

6. (1 pont) Írj Pascal vagy C/C++ **eljárást** (void-függvényt), amely paraméterként megkapja az *n* nullánál nagyobb, *legtöbb négyszámjegyű* természetes számot, majd pontosan *ötször* új számot képez úgy, hogy minden lépésben kiszámítja a kurrens szám számjegyei köbének összegét, és az így kapott érték lesz a következő szám. Az eljárás írja ki az eredeti számot, valamint a generált számsorozatot is, az elemeket egy-egy szóközzel elválasztva.

Példa:

Bemenet:

```
12
```

Kimenet:

```
12 9 729 1080 513 153
```

Magyarázat: $1^3 + 2^3 = 9$, $9^3 = 729$, $7^3 + 2^3 + 9^3 = 1080$, stb.

A függvény egy lehetséges fejléce C-ben, az alábbi:

```
void szamsor ( int n ){
```

```
            …
        }

void szamsor( int n ) {
    cout << n << " "; // eredeti szám
    for (int i = 0; i < 5; i++) {
        int szam = n;
        int osszeg = 0;
        // számjegyek köbének összege
        while (szam > 0) {
            int szj = szam % 10;
            osszeg += szj * szj * szj;
            szam /= 10;
        }
        n = osszeg;
        cout << n << " ";
    }
}
```

7. (1 pont) Írj egy C/C++ vagy Pascal **programot**, amely beolvassa a *matrix.in* állományból a *n*×m méretű ($2 \le n, m \le 100$), pozitív egészeket tartalmazó mátrixot. Számoljuk meg, hogy hány lokális maximumot tartalmaz a mátrix, majd írassuk ki a képernyőre. Egy számot lokális maximumnak nevezünk, ha szigorúan nagyobb, mint a közvetlen szomszédjai fel, le, bal és jobb irányban. A bemenet első sora két egész számot tartalmaz szóközökkel elválasztva (*n*, *m*). A következő *n* sor mindegyike *m* darab nullánál nagyobb természetes számot tartalmaz, ugyancsak szóközökkel elválasztva.

Példa:

Bemenet:

```
5 5
1 2 3 2 1
4 9 5 9 2
3 2 8 2 3
2 5 3 6 2
1 2 1 2 1
```

Kimenet:

```
6
```

(A lokális maximumokat bejelöltük)

```
const int MAX = 100;
int main() {
    int n, m;
    int matrix[MAX][MAX];
    ifstream fin("matrix.in");
    fin >> n >> m;
    // Beolvasás
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            fin >> matrix[i][j];
    fin.close();
    int lokalisMaxCount = 0;
    // Lokális maximum keresése (csak 4 irány: fel, le, bal, jobb)
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            int szam = matrix[i][j];
            bool nagyobbMindnel = true;
            // bal
            if (j > 0 && matrix[i][j - 1] >= szam)
                nagyobbMindnel = false;
            // jobb
            if (j < m - 1 && matrix[i][j + 1] >= szam)
```

```
                nagyobbMindnel = false;
            // fel
            if (i > 0 && matrix[i - 1][j] >= szam)
                nagyobbMindnel = false;
            // le
            if (i < n - 1 && matrix[i + 1][j] >= szam)
                nagyobbMindnel = false;
            if (nagyobbMindnel)
                ++lokalisMaxCount;
        }
    }
    cout << lokalisMaxCount << endl;
    return 0;
}
```

8. (1 pont) Írj Pascal vagy C/C++ **eljárást (void-függvényt)**, amely paraméterként megkapja az *n* és *m* természetes számokat ($1 \le n, m \le 1000$, *m* páratlan), valamint egy kétdimenziós tömböt (legyen *a*), amely egy *n*×*m* méretű mátrixot. Az eljárás írja a ki képernyőre a mátrix oszlopainak bizonyos elemeit (egy sorban, szóközzel elválasztva) az alábbi szabály szerint:

- az 1. oszlopnak írassuk ki minden elemét: 1., 2., 3., …;
- a 2. oszlopnak írassuk ki minden második elemét: 1., 3., 5., ...;
- a 3. oszlopnak írassuk ki minden harmadik elemét: 1., 4., 7., ...;
- így haladunk a középső oszlopig, mondjuk a *k*. oszlopig, ahol minden *k*-adik elemet írassunk ki;
- a középső oszlop utánitól, egészen az utolsó oszlopig, csökkenő távolságokat alkalmazunk az oszlopok elemeinek kiíratásakor;
- a (k+1). oszlopnak írassuk ki minden (k-1). elemét;
- a (k+2). oszlopnak írassuk ki minden (k-2). elemét;
- és így tovább, egészen az utolsó oszlopig, amelynek újra minden elemét írassuk ki.

A függvény egy lehetséges fejléce C-ben, az alábbi:

```
void mintazat ( int a[][1000], int n, int m ){
      …
}
```

Példa:

Amennyiben *n*=5, *m*=5, és az *a* tömb az alábbi mátrixot tárolja:

1	**2**	**3**	**4**	**5**
6	7	8	9	**10**
11	**12**	13	**14**	**15**
16	17	**18**	19	**20**
21	**22**	23	**24**	**25**

A képernyőn megjelenítendő számsor a következő:

```
1 6 11 16 21 2 12 22 3 18 4 14 24 5 10 15 20 25
```

(Kiemeltük a tömb kiírandó elemeit)

```
void mintazat( int a[][1000] , int n , int m ) {
    int kozep = m / 2 + 1;
    for (int j = 0; j < m; ++j) {
        int lepes;
        if (j < kozep) {
            lepes = j + 1; // bal oldal: 1., 2., 3., ..., k-adik
        } else {
            lepes = m - j; // jobb oldal: k-1, k-2, ..., 1
        }
        for (int i = 0; i < n; i += lepes) {
            cout << a[i][j] << " ";
        }
```

```
    }
}
```

9. (1 pont) Írj egy hatékony (O(log *n*) időbonyolultságú) **keresőfüggvényt** (Pascal vagy C/C++), amely paraméterként kap egy *rendezett* (növekvő vagy csökkenő sorrendű), egész számokat tartalmazó tömböt (legyen *a*), a tömb elemszámát (legyen *n*), valamint egy egész értéket, amelyet keresni kell a sorozatban (legyen *x*). A függvény térjen vissza a keresett érték pozíciójával (indexével) a számsorozatban. Ha a keresett elem többször is előfordul, a függvény adja vissza az előfordulási szakasz középső elemének indexét. Ha a keresett érték nem szerepel a sorozatban, akkor a visszatérési érték legyen `-1`.

A függvény egy lehetséges fejléce C-ben, az alábbi:

```
int keres ( int a[], int n, int x ){
      …
}
```

Példa

Ha a rendezett sorozat [3,3,2,2,2,2,2,**2**,2,2,2,2,2,2,1], a keresett érték pedig `2`, akkor az eredmény `8`.

```
int keres(int a[], int n, int x) {
    // Eldöntjük, növekvő vagy csökkenő sorrendű a tömb
    bool novekvo = (n >= 2) ? (a[0] < a[n - 1]) : true;
    // Bináris keresés a bal és jobb határ megtalálásához
    int bal = -1, jobb = -1;
    // BAL (legelső előfordulás)
    int l = 0, r = n - 1;
    while (l <= r) {
        int m = (l + r) / 2;
        if (a[m] == x) {
            bal = m;
            r = m - 1;
        } else if ((novekvo && a[m] < x) || (!novekvo && a[m] > x)) {
            l = m + 1;
        } else {
            r = m - 1;
        }
    }
    // JOBB (utolsó előfordulás)
    l = 0, r = n - 1;
    while (l <= r) {
        int m = (l + r) / 2;
        if (a[m] == x) {
            jobb = m;
            l = m + 1;
        } else if ((novekvo && a[m] < x) || (!novekvo && a[m] > x)) {
            l = m + 1;
        } else {
            r = m - 1;
        }
    }
    if (bal == -1 || jobb == -1)
        return -1; // nem található
    return (bal + jobb) / 2; // középső előfordulás
}
```