



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 programokat megjegyzésekkel kell ellátni!**
- **Törekedjünk hatékony megoldásokra!** (maximális pontszám feltétele)

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

```
res = 0
amíg n ≠ 0 végezd
|   res = res + (n % 10) * 2
|   n = n / 10
kiír res
```

Mit ír ki a fenti programrészlet, ha $n = 20260907$? 52

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

```
res = 0
amíg n ≠ 0 végezd
|   ha (n % 10) % 3 == 0 akkor
|   |   res = res + n % 10
|   n = n / 10
kiír res
```

Mit ír ki a fenti programrészlet, ha $n = 20260907$? 15

3. (1 pont) Legyen az alábbi pszeudokód programrészlet:

```
res = 1
amíg num ≠ 0 végezd
|   ha num % 10 > 5 akkor
|   |   res = res * (num % 10)
|   különben
|   |   res = res + (num % 10)
|   num = num / 10
kiír res
```

Mit ír ki a fenti programrészlet, ha $num = 2026$? 10

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

```
res = 0
i = num % 10
amíg num ≠ 0 végezd
|   x = num % 10
|   ha x > i akkor
|   |   res = res + 1
|   i = x
|   num = num / 10
kiír res
```

Mit ír ki a fenti programrészlet, ha $num = 2026$? 1



5. (1 pont) Írj Pascal vagy C/C++ **programot**, amely beolvas a billentyűzetről három egész számot (ezek egy háromszög szögeinek mértékét jelentik, fokban kifejezve), majd eldönti és kiírja, hogy a megadott szögek derékszögű háromszöget alkotnak-e. Ha igen, akkor írd ki a képernyőre az „IGEN”, máskülönben a „NEM” üzenetet.

```
#include <iostream>
using namespace std;

int main() {
    int a, b, c;
    cin >> a >> b >> c;

    // háromszög: a szögek összege 180, derékszögű: az egyik szög 90 fok
    if (a + b + c == 180 && (a == 90 || b == 90 || c == 90)) {
        cout << "IGEN" << endl;
    } else {
        cout << "NEM" << endl;
    }

    return 0;
}
```

6. (1 pont) Írj Pascal vagy C/C++ **programot**, amely beolvas a billentyűzetről egy n ($1 \leq n \leq 1000$) természetes számot, egy k ($2 \leq k \leq 100$) természetes számot, majd egy n elemű, egész számokat tartalmazó számsorozatot. Számold ki, hogy hány szám van a tömbben, amely a k szám többszöröse, és írasd ki a képernyőre! Példák:

Bemenet:

5 3
6 33 4 9 1

Kimenet:

3

Bemenet:

4 9
4 2 3 3

Kimenet:

0

```
#include <iostream>
using namespace std;

int main() {
    int n, k;
    cin >> n >> k;

    int szam;
    int darab = 0;

    // minden számot beolvasunk és azonnal ellenőrizzük, így nem kell tömböt használni
    for (int i = 0; i < n; ++i) {
        cin >> szam;
        if (szam % k == 0) { // ha a szám osztható k-val
            darab++;
        }
    }

    cout << darab << endl; // kiírjuk az eredményt
    return 0;
}
```



7. (1 pont) Írj Pascal vagy C/C++ **eljárást** (void-függvényt), amely paraméterként megkap egy egész számokat tartalmazó mátrixot és a mátrix n, m méreteit ($1 \leq n \leq 100, 1 \leq m \leq 100$). Az eljárás lenullázza az adott mátrix minden olyan sorát, amely csupán páros számokat tartalmaz. Példa:

Eredeti mátrix:

1	2	3	4	5
6	8	10	12	14
2	1	4	3	6

Végső mátrix:

1	2	3	4	5
0	0	0	0	0
2	1	4	3	6

```
// Az eljárás paraméterként kapja a mátrixot és a méreteit (n sor, m oszlop)
void nullazParosSorok(int a[100][100], int n, int m) {
    for (int i = 0; i < n; ++i) {
        // megnézzük, hogy a sor minden eleme páros-e
        bool mindParos = true;

        for (int j = 0; j < m; ++j) {
            if (a[i][j] % 2 != 0) { // ha páratlan számot találunk
                mindParos = false;
                break; // nem kell tovább vizsgálni a sort
            }
        }

        // ha a sor csak páros számokat tartalmaz, lenullázzuk
        if (mindParos) {
            for (int j = 0; j < m; ++j) {
                a[i][j] = 0;
            }
        }
    }
}
```

8. (1 pont) Írj Pascal vagy C/C++ **programot**, amely egy bemeneti állományból (*input.txt*) beolvassa az n értéket (ahol $2 \leq n \leq 50$) és egy $n \times n$ természetes számokat tartalmazó mátrixot. A programnak el kell döntenie, hogy hány különálló 1-s folt található a mátrixban. Egy 1-es akkor van összeköttetésben egy másikkal, ha fel/le/jobbra/balra található tőle. A mátrixban a 0-s értékek jelentik a foltok közötti határvonalakat. A kimenetet írassuk ki a képernyőre.

Példa:

Bemenet:

5				
0	0	1	1	1
0	1	1	0	0
1	1	1	0	1
0	0	0	1	1
1	1	1	1	1

Kimenet:

2

```
// Megszámoljuk az egymással összeköttetésben lévő 1-es foltokat.
// Módszer: BFS (iteratívan). Minden új 1-est találva megnöveljük a
// foltok számát, majd bejárjuk a folt összes szomszédos 1-esét,
// és azokat 0-ra állítjuk, hogy ne számoljuk a foltokat újra.

#include <iostream>
#include <fstream>
#include <stack>
#include <utility>

using namespace std;

int main() {
    ifstream fin("input.txt"); // beolvasás input.txt-ből
```



```
int n;
int a[51][51];

fin >> n;

// beolvassuk a mátrixot
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        fin >> a[i][j];
    }
}

int foltok = 0; // a foltok száma
stack<pair<int, int>> verem; // a bejáráshoz használt verem

// végigjárjuk a mátrixot
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        if (a[i][j] != 1) {
            continue; // ha nem 1-es, akkor folytatjuk a következő cellával
        }

        foltok++; // a foltok száma eggyel nő
        verem.push({i, j}); // a kiinduló cella a verembe kerül

        // a verem segítségével bejárjuk a folt összes 1-esét
        while (!verem.empty()) {
            pair<int, int> akt = verem.top();
            verem.pop();

            int x = akt.first;
            int y = akt.second;
            a[x][y] = 0; // megjelöljük, hogy már bejártuk (0-ra)

            // fel
            if (x > 0 && a[x - 1][y] == 1) {
                a[x - 1][y] = 0;
                verem.push({x - 1, y});
            }
            // le
            if (x < n - 1 && a[x + 1][y] == 1) {
                a[x + 1][y] = 0;
                verem.push({x + 1, y});
            }
            // balra
            if (y > 0 && a[x][y - 1] == 1) {
                a[x][y - 1] = 0;
                verem.push({x, y - 1});
            }
            // jobbra
            if (y < n - 1 && a[x][y + 1] == 1) {
                a[x][y + 1] = 0;
                verem.push({x, y + 1});
            }
        }
    }
}

cout << foltok << endl; // kiírjuk a foltok számát
return 0;
}
```



9. (1 pont) Írj Pascal vagy C/C++ **függvényt**, amely egy n méretű számjegy sorozatot megvizsgál, hogy az elemeit át lehet-e úgy rendezni, hogy palindrom számjegy sorozatot kapj belőle. Egy sorozat akkor palindrom, hogy ha visszafelé olvasva is ugyanazt a sorozatot kapod. A függvény visszatérési értéke 1, ha lehet a számjegy sorozatból palindrom sorozatot alkotni, máskülönben -1, ha nem.

Példák:

Bemenet:

5
9 4 9 4 1

Kimenet:

1

Bemenet:

7
8 6 0 4 2 9 4

Kimenet:

-1

```
// Egy számjegy-sorozat pontosan akkor rendezhető át palindrommá,  
// ha legfeljebb egy olyan számjegy van, amely páratlan sokszor fordul elő  
// (az páros sokszor előforduló számjegyeket a palindrom két oldalán  
// szimmetrikusan elhelyezhetjük, a páratlan kerül a közepére).  
// Visszaad 1-et, ha a számjegyekből palindrom rendezhető, különben -1-et  
int lehetPalindrom(int szamjegyek[], int n) {  
    int darab[10] = {0}; // darab[d] = hányszor fordul elő a d számjegy (0-9)  
  
    // megszámoljuk, melyik számjegy hányszor szerepel  
    for (int i = 0; i < n; ++i) {  
        darab[szamjegyek[i]]++;  
    }  
  
    int paratlanok = 0; // hány számjegy fordul elő páratlan sokszor  
  
    for (int i = 0; i < 10; ++i) {  
        if (darab[i] % 2 != 0) {  
            paratlanok++;  
        }  
    }  
  
    // palindrom csak akkor lehetséges, ha legfeljebb egy páratlan számjegy van  
    if (paratlanok <= 1) {  
        return 1;  
    } else {  
        return -1;  
    }  
}
```