IPAR 4.0 C# .NET ALAPOKON

7. forduló

SEMILAB

A kategória támogatója: Semilab Zrt.

RENDELKEZÉSRE ÁLLÓ IDŐ:

60:00

Ismertető a feladathoz

A gyár további folymatok elvégzéséhez kéri a segítségünket, részletes leírások a feladatokban.

Felhasznált idő: 02:07/60:00 **Elért pontszám:** 0/29

1. feladat 0/5 pont

Mit ír ki az alábbi kódrészlet (végtelen futási időt engedélyezve)?

```
0 references
private static void Main(string[] args)
{
    long sum = 0;
    for (long i = 0; i < 999_000_000_999; i++)
    {
        sum += i % 100;
        sum = sum % 100;
    }
    Console.WriteLine(sum);
}
```

Válaszok

A helyes válasz:

1

Magyarázat

Mivel a sum változó módosítása csak (i % 100)-tól függ, ezért 100-asával ismétlődik a módosítás.

Könnyű látni, hogy ha i értéke 0-99-ig megy, akkor 50-nel növekszik a sum értéke (párba rakva a számokat: 1-99, 2-98, ..., 49-51 a párok együttes hatása 0 lesz, így csak a párosításból kimaradt 50 számít).

A modulo 100 miatt a következőképpen alakul sum értéke az i ciklusváltozó max értéke függvényében:

99: 50

199: 0

299: 50

399: 0

...

Látszik, hogy minden páros százas helyértéknél a sum 50, páratlan esetben 0. Ezért 999 000 000 899: **50**

999 000 000 900-tól 999 000 000 998-ig a sum értéke 51-gyel növekszik (2-98, 3-97, 49-51 végű párok kiütik egymást, kimaradt az 1 és az 50 végű -> **51**)

Így a legvégén a sum értéke (**50** + **51**) % 100 = **1** lesz.

2. feladat 0/12 pont

Egy gyárban a mérőgépünk többféle módon is képes megmérni a wafert.

Bizonyos méréseknek csak akkor kell lefutnia, ha másik mérés eredménye ezt megkívánja.

A mérések a következők:

- *'A'* típusú mérés: Első mérés, mindenképp lefut, és true/false eredményt ad egy waferre. Végrehajtási ideje 20 másodperc.
- *'B'* típusú mérés: Ha az 'A' true eredménnyel tért vissza, akkor fut le. Végrehajtási ideje 30 másodperc.
- *'C'* típusú mérés: Ha az 'A' false eredménnyel tért vissza, akkor fut le. Végrehajtási ideje 50 másodperc.

A különböző típusú mérések futhatnak párhuzamosan. A waferek futószalagon egymás után jönnek, azonnal mérjük őket, amint tudjuk.

A **7_fordulo_2_feladat.txt** tartalma egy-egy waferhez az 'A' mérés eredménye. Az 1 jelöli a true, 0 jelöli a false értéket. Adjuk meg, hogy összesen mennyi idő alatt mérjük le az input fileban felsorolt wafereket ('A' mérés idejét is beleszámítva).

Teszteléshez *7_fordulo_2_feladat_test.txt* eredménye: 2600

Válaszok

A helyes válasz:
2540

Magyarázat

Folyamatosan tároljuk, hogy az egyes mérések mikor végeznének az egyes minták beérkezése után. Egy-egy új minta esetén az "A" végzési ideje mindig a saját befejezési ideje + egy új "A" mérés ideje. Az új mintára "B" mérést akkor tudjuk elkezdeni, ha B és A is befejezte az addig feldolgozandó műveleteket, ezért max(A,B)-hez adjuk hozzá az új minta "B" feldolgozási idejét. Ugyanez igaz C-re. Akkor fejezzük be az összes elem feldolgozását, ha az összes mérés befejeződik, ezért vesszük az összes mérés maximumát. (Mivel A-t mindenképp követi B vagy C, elég csak max(B,C))

```
private static int MetrologyAllTime(string inputFile)
{
    const int A_TIME = 20;
    const int B_TIME = 30;
    const int C_TIME = 50;

    List<bool> resultAList = File.ReadAllText(inputFile)
                            .Split(',')
                            .Select(int.Parse)
                            .Select(item => item == 1 ? true : false)
                            .ToList();
    int finishA = 0;
    int finishB = 0;
    int finishC = 0;

    foreach (var resultA in resultAList)
    {
        finishA += A_TIME;

        if (resultA)
        {
            finishB = Math.Max(finishA, finishB) + B_TIME;
        }
        else
        {
            finishC = Math.Max(finishA, finishC) + C_TIME;
        }
    }

    return Math.Max(finishB, finishC);
}
```

3. feladat 0/12 pont

A gyártó a CPU-kat egy négyzetes spirálban helyezi el a waferen, ahol az első CPU a bal felső sarokba kerül, az őt követő pedig tőle jobbra helyezkedik el.

Ezt az elrendezést szemlélteti a CPUPositions(int size) függvény, amely megadja az egyes CPU-k pozícióját. A függvény size^2 CPU-t helyez el négyzetes spirálban, az első CPU-t 1-essel jelölve.

> CPUPositions(3)

Kimenet:

1 2 3

8 9 4

7 6 5

> CPUPositions(6)

Kimenet:

1 2 3 4 5 6

20 21 22 23 24 7

19 32 33 34 25 8

18 31 36 35 26 9

17 30 29 28 27 10

16 15 14 13 12 11

Készítse el a CPUPositionsInRow(int size, int rowIdx) függvényt, amely a CPUPositions által bemutatott elrendezés rowIdx-edik sorában lévő pozíciókat adja vissza ','-vel elválasztva (a sorok indexelése a 0. sortól kezdődik).

Például:

> CPUPositionsInRow (6, 4)

Kimenet:

17,30,29,28,27,10

Adjuk meg a **CPUPositionsInRow (21, 10)** eredményét! (Szóköz nélkül vesszővel elválasztott számok)

Válaszok

A helyes válasz:
71,144,209,266,315,356,389,414,431,440,441,436,423,402,373,336,291,238,177,108,31

Magyarázat

Spirál szerűen megyünk végig a tömbön folyamatosan tárolva minden irányban a spirál határát. Miután felépítettük a mátrixot, kiszedjük a keresett sort belőle.

```
private static int[] CPUPositionsInRow(int size, int rowIndex)
{
    int[,] matrix = new int[size, size];
    int topIndex = 0;
    int bottomIndex = size-1;
    int leftIndex = 0;
    int rightIndex = size - 1;
```

```
    int count = 1;
    while (count <= size * size)
    {
        for (int i = leftIndex; i <= rightIndex; i++)
        {
            matrix[topIndex, i] = count++;
        }
        topIndex++;

        for (int i = topIndex; i <= bottomIndex; i++)
        {
            matrix[i, rightIndex] = count++;
        }
        rightIndex--;

        for (int i = rightIndex; i >= leftIndex; i--)
        {
            matrix[bottomIndex, i] = count++;
        }
        bottomIndex--;

        for (int i = bottomIndex; i >= topIndex; i--)
        {
            matrix[i, leftIndex] = count++;
        }
        leftIndex++;
    }

    var row = new int[size];
    for (int i = 0; i < size; i++)
    {
        row[i] = matrix[rowIndex, i];
    }

    return row;
}
```