C# feladatgyűjtemény

Kovács Emőd, Radványi Tibor, Király Roland, Hernyák Zoltán

Eszterházy Károly Főiskola, Matematikai és Informatikai Intézet

A tananyag a TÁMOP-4.1.2-08/1/A-2009-0046 számú Kelet-magyarországi Informatika Tananyag Tárház projekt keretében készült. A tananyagfejlesztés az Európai Unió támogatásával és az Európai Szociális Alap társfinanszírozásával valósult meg.

Nemzeti Fejlesztési Ügynökség http://ujszechenyiterv.gov.hu/ *06 40 638-638*

2011

Tartalom

1. fejezet - Előszó

Ez a feladatgyűjtemény a C# tankönyv című jegyzetet egészíti ki, segítve ezzel a kedves olvasót abban, hogy minél jobban elsajátíthassa a nyelv jellemzőit és használatát.

Az jegyzet első néhány fejezetében szereplő feladatok nagy részének a megoldását, valamint a kimeneti képernyő képét is közöltük. Feltételeztük, hogy az első néhány fejezet feladataival próbálkozó kedves olvasó még nem jártas a C# programozási nyelv használatában, és ezen okból kifolyólag az egyszerűbb programok írása nehézségeket okozhat a számára.

Hasonló okokból a bonyolultabb feladatokat, valamint a kevésbé közismert fogalmakat, vagy éppen a matematikai formulákat tartalmazó részeket magyarázatokkal láttuk el, azok egyszerűbb feldolgozása érdekében. A fejezetekben található programok megoldásait a fejezetek végén helyeztük el (kivételt képeznek ez alól azok a feladatok, ahol a forrásszöveget szétválasztva a leírástól, a feladat szövege nem lenne értelmezhető).

Mindezek mellett számos feladat szövege tartalmaz érdekes, valamint mindenki számára hasznos információkat az adott problémával, vagy a benne szereplő ismeretekkel kapcsolatban azért, hogy színesebbé tegye azokat, valamint érdekesebbé varázsolja a C# programozási nyelv elsajátításának a folyamatát.

2. fejezet - Az adatok be és kivitele, és az elágazások (szerző: Király Roland)

Tartalom

2.1. feladat (Kezdetek - Hello World – *szint: 1*). Mielőtt bonyolultabb programok írásába kezdenénk, készítsük el a manapság már klasszikusnak számító „Hello Világ” programot.

Magyarázat: A program elkészítéséhez használjuk a *Console* osztály *WriteLine* metódusát, melyet a *Console Program* osztály *main* nevű metódusában helyezhetünk el.

```
Console.WriteLine("Hello világ");
```

Információ: Egyesek sportot űznek abból, hogy megpróbálják a Hello Világ programot a fellelhető összes programozási nyelven megírni. Ezt a tevékenységüket dokumentálják is egy weboldalon, ahol találhatunk egy soros, és több oldalas programok is. A *C++* megoldás, amely alapján a C# verzió gyorsan elkészíthető, az alábbi listában látható.

```
#include <iostream>;

int main()
{
  std::cout << "Hello World!" << std::endl;
  return 0;
}
```

Az Interneten, rövid keresés után bizonyosan találhatunk a *C++* nyelvi verziónál jóval hosszabb, vagy éppen extrémebb megoldásokat is.

2.1. ábra. A Hello World feladat megoldása

A 2.1 program bemutatja a feladat egy lehetséges megoldását, amelyet kedvünk szerint továbbfejleszthetünk. A megoldás kimeneti képernyőjét a 2.1 ábrán tekinthetjük meg.

2.2. feladat (Számok bekérése – *szint: 1*). Írjunk programot, mely bekér egy számot, és eldönti, hogy osztható-e 3-mal, 4-gyel vagy 9-cel.

A 2.2 program bemutatja az oszthatósági feladat egy megoldását. Amennyiben elég erőt érzünk magunkban, próbáljuk meg a programot kevesebb programsorral megoldani!

2.3. feladat (Átváltások – *szint: 1*). Készítsünk programot, mely bekér egy hőmérséklet értéket, majd felajánlja, hogy Celsiusból Fahrenheitbe, vagy Fahrenheitből Celsiusba váltja át.

A 2.3 program bemutatja a fokból fahrenheitbe átváltó feladat egyszerű megoldását. Az átváltáshoz használjuk a következő összefüggést: $1°\mathcal{C} = \mathcal{K},\ \text{valamint}\ 1\mathcal{K} = 1.8\mathcal{F}$ Készítsük el a programot úgy, hogy az több információt közöljön a felhasználóval arra nézve, hogy valójában mire képes!

2.4. feladat (Testtömeg indexek – *szint: 2*). Írjunk programot, mely a testsúly és a testmagasság alapján meghatározza a testtömegindexet, és kiírja, hogy milyen testsúly osztályba tartozik az adott illető. a testtömeg osztályokat meghatározhatjuk tetszőlegesen, de alapul vehetünk létező osztályozásokat is.

$$Testtomegindex = Testtomeg[kg]/testmagassg^2[m^2]$$

A 2.4 programban megtalálhatjuk a testtömeg indexet kiszámító feladat megoldását, amit IF-THEN-ELSE elágazások helyett elkészíthetünk switch, vagy lista segítségével.

2.5. feladat (Víz-gőz-jég – *szint: 1*). Készítsünk programot, amely bekéri a víz hőmérsékletét, majd eldönti, hogy az milyen halmazállapotú. A halmazállapot lehet folyékony, gőz, vagy jég.

A 2.5 forrásszövegben megtekinthetjük víz halmazállapotát előállító programot. Mivel a program elég rövid, a gyakorlás kedvéért próbáljuk meg színekkel érdekesebbé tenni a konzol kimenetet!

2.6. feladat (Pontok távolsága – *szint: 2*). Írjunk programot, amely bekéri két pont koordinátáit, majd kiszámolja azok távolságát.

Magyarázat:

A távolság a két pont közé eső szakasz hossza, melyet a pontok koordinátáiból könnyedén kiszámolhatunk.

$$\sqrt{(x1-x2)*(x1-x2)+(y2-y1)*(y2-y1)}$$

A 2.6 forrásszövegben megtekinthetjük pontok távolságát kiszámító program megoldását. Mivel ez a program is elég rövid, a gyakorlás kedvéért próbáljuk meg színekkel érdekesebbé tenni a konzol kimenetet!

2.7. feladat (Ponthatárok – *szint: 2*). Írjon egy programot, ami leosztályoz egy maximálisan 100 pontos dolgozatot az 50, 65, 80, 90 ponthatárok szerint! A határérték a jobb jegyhez tartozik. Ha a pontszám negatív vagy száznál nagyobb, akkor a program írja ki, hogy hibás az adat!

A 2.7 forrásszövegben találjuk meg a dolgozatok osztályozását végző feladat megoldását. A sok IF helyett itt is megpróbálhatunk *switch* típusú elágazást alkalmazni.

2.8. feladat (Mezőgazdasági jóslás – *szint: 1*). Készítsen konzolos alkalmazást, amely mezőgazdasági jóslást végez. A program kérje be az elvetett búza mennyiségét tonnában. Ez alapján számolja ki egy véletlenszerűen generált szorzóval (5-15) a várható hozamot, és írja ki a mennyiségét. A szorzó alapján elemezze és írja ki, hogy milyen év várható: átlag alatti (5-8), átlagos év (9-12), átlag feletti (13-15).

A 2.8 programszövegben találjuk meg a feladat megoldásának a lehető legegyszerűbb változatát, amelyet természetesen továbbfejleszthetünk.

2.9. feladat (Respirációs kvóciens kiszámítása – *szint: 3*). Készítsünk az egészség megőrzéséhez használható programot. A programunk kérje be a kilégzéskor keletkező CO2 és O2 mennyiségét! Számoljuk ki a respirációs kvócienst!

Magyarázat: Az anyagcsere folyamán a keletkezett CO^2 és a felhasznált O^2 hányadosa, vagyis a légzési hányados. (RQ = kilégzett CO^2. Belégzett O^2 aránya). Az értékének a kiszámításához használhatjuk a következő képletet: $RQ = CO^2/O^2$. Az RQ akkor megfelelő, ha értéke 0,8-as értéket mutat. Ha ennél kevesebb, akkor a szervezet a zsírokból nyeri az energiát. Ha ennél több, akkor a szénhidrátokból.

2.10. feladat (Igazolatlan hiányzások – *szint: 1*). Készítsünk programot, amely beolvassa egy diák igazolatlan hiányzásainak számát. Ennek megfelelően írassuk ki a magatartás jegyét. Tíz igazolatlan hiányzás elérésekor (vagy ha ezt túlhaladtuk) kérjük be a tanuló születési dátumát és írjuk ki az igazolatlan hiányzásait (amennyiben az érték több mint tíz). Készítsünk kategóriákat az igazolatlan hiányzások száma alapján. Az első kategória figyelmeztetést, a második osztályfőnöki intőt, a harmadik igazgatói megrovást, a negyedik kategória pedig felfüggesztést von maga után. A büntetés mértékét szintén jelezzük a felhasználó felé.

2.11. feladat (Véletlen számok listája – *szint: 1*). Készítsünk programot, amely bekér két számot, majd a kettő közötti számtartományban kiír három darab véletlen számot.

A 2.9 programszövegben megtaláljuk a feladat egy lehetséges megoldását. Amennyiben háromnál több véletlen számot kell előállítani, készítsük el a ciklussal működő változatot! Ehhez természetesen meg kell ismerkednünk a ciklus utasítás valamelyik változatával.

2.12. feladat (Pénzérmék – *szint: 1*). Készítsünk programot, amely bekér egy összeget, majd kiírja, hogy azt hogyan lehet a lehető legkevesebb pénzérméből összeállítani.

Magyarázat: A program valójában egy címletező program, mely hasonlóan működik, mint a számrendszerekbe történő átváltások, azzal a kivétellel, hogy ebben az esetben nem a számrendszer alapszámával, hanem mindig a megfelelő címlettel kell osztanunk mindaddig, amíg el nem fogy az összeg.

Ha ügyesek vagyunk, megpróbálhatjuk előállítani az összes lehetséges megoldást, vagyis az adott összeg összes lehetséges felosztását a címletek alapján.

2.13. feladat (Csomagoló cég programja. – *szint: 2*). Készítsünk programot, amely dinnyék csomagolásához végez számításokat. A dinnyéket szalaggal kell átkötni úgy, hogy kétszer körbe érje őket, és a masni készítéséhez számolunk még 60 cm-t. A program kérje be a dinnye átmérőjét, és a dinnyék számát! Számítsa ki, és írja a képernyőre, hogy *n* dinnye csomagolásához hány méter szalagra van szükség.

A 2.10 forrásban megtaláljuk a dinnye csomagoló program megoldását, amely használja a Math osztályban implementált Pi értéket. Helyette használhatnánk a 3.14-es értéket. Mi változna ekkor?

2.14. feladat (Csempézés – *szint: 1*). Készítsünk programot, amely segíti a burkoló mesterek munkáját. A szükséges csempe mennyiségének a kiszámításához a program kérje be a terület szélességét, valamint a magasságát méterben, majd számolja ki, hogy 20cm*20cm méretű csempék esetén hány darabra van szükség a munka elvégzéséhez (a plusz 10%-ot az illesztések miatt illik rászámolnunk). A 2.11 forrásszövegben találjuk a megoldást.

2.15. feladat (Sokszögek – *szint: 2*). készítsünk programot, amely kiszámolja sokszögek átlóit. Az adatok bekérése után (szabályos háromszög, négyszög, ötszög, hatszög oldalai, valamint azok magassága) kiszámolja az átlók hosszát.

2.16. feladat (Sokszögek és körök – *szint: 2*). készítsünk programot, amely kiszámolja sokszögek átlóit. Az adatok bekérése után (szabályos háromszög, négyszög, ötszög, hatszög oldalai, valamint azok magassága) kiszámolja a beírható, és a körülírt körök sugarát.

2.17. feladat (Percek és órák – *szint: 1*). Készítsünk programot, amely bekér két, egy napon belüli időpontot (óra, perc, másodperc formátumban. Számítsuk ki a két időpont közti különbséget másodpercekben és írassuk ki a képernyőre!

2.18. feladat (Másodfokú egyenlet – *szint: 3*). Kérjük be a másodfokú egyenlet együtthatóit a,b,c, majd írjuk ki, hogy hány valós gyöke van az egyenletnek!

$$ax^2 + bx + c = 0$$

A 2.12 forrásszövegben találjuk a feladat egy nem túl kifinomult megoldását ami arra mindenképpen jó lesz, hogy ez alapján jobbat készíthessünk.

A fejezet forráskódjai

2.1. forráskód. A Hello World feladat megoldása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace kezdetek
{
      class Program
      {
            static void Main(string[] args)
            {
             Console.WriteLine("Hello Világ!");
             Console.ReadLine();
            }
      }
}
```

2.2. forráskód. Az oszthatósági feladat megoldása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ProgNyelvek
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("Kérek egy számot: ");
         string n = Console.ReadLine();
         Console.WriteLine();
         int hossz=n.Length;
         int osszeg = 0;
         for (int i = 0; i < hossz; i++)
            osszeg = osszeg+Convert.ToInt16(n[i])-48;
         if (osszeg % 3 == 0)
            Console.WriteLine("A szám osztható 3-mal.");
            else Console.WriteLine("A szám nem osztható 3-mal.");
         if (osszeg % 9 == 0)
            Console.WriteLine("A szám osztható 9-cel.");
            else Console.WriteLine("A szám nem osztható 9-cel.");
         if (hossz>1)
         {
            if ((Convert.ToInt16(n[hossz-2]-48)*10+
```

```
            Convert.ToInt16(n[hossz-1])-48) % 4 == 0)
             Console.WriteLine("A szám osztható 4-gyel.");
            else Console.WriteLine("A szám nem osztható 4-gyel.");
         }
         else if ((Convert.ToInt16(n[0])-48) % 4 == 0)
         Console.WriteLine("A szám osztható 4-gyel.");
         else Console.WriteLine("A szám nem osztható 4-gyel.");
         Console.ReadLine();
      }
   }
}
```

2.3. forráskód. A hőmérséklet vizsgáló feladat megoldása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace _1b
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("Adj meg egy hőmérséklet értéket: ");
         int n = int.Parse(Console.ReadLine());
         Console.Write
             ("Válassz opciót: (1) C° --> K° (2) K° --> C° : ");
         byte c = byte.Parse(Console.ReadLine());
         Console.WriteLine();
         switch (c)
         {
            case 1:
               Console.WriteLine("{0} C° = {1} K°", n,n+273);
               break;
            case 2:
               Console.WriteLine("{0} K° = {1} C°", n,n-273);
               break;
         }
         Console.ReadLine();
      }
   }
}
```

2.4. forráskód. A testtömeg indexet kiszámító feladat megoldása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ttindex
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("Testtömeg[kg]: ");
         int m = int.Parse(Console.ReadLine());
         Console.Write("Testmagasság[cm]: ");
         double h = double.Parse(Console.ReadLine());
         h = h / 100;
         double tti = m / Math.Pow(h, 2);
         Console.WriteLine("Testtömegindex: {0}", tti);
         Console.Write("Testsúlyosztály: ");
         if (tti < 16) Console.WriteLine("Súlyos sovány ság");
         else if (tti < 17) Console.WriteLine("Mérsékelt sovány ság");
         else if (tti < 18.5) Console.WriteLine("Enyhe sovány ság");
         else if (tti < 25) Console.WriteLine("Normális testsúly");
         else if (tti < 30) Console.WriteLine("Túlsúlyos");
         else Console.WriteLine("Elhízás");
         Console.ReadLine();
      }
   }
}
```

2.5. forráskód. A víz halmazállapotát felismerő program forráskódja

```
static void Main(string[] args)
{
   Console.WriteLine("A viz halmazallapotanak vizsgalata:");
   Console.Write("Homerseklet: ");

   double t= Convert.ToDouble(Console.ReadLine());

   if (t > 0)
   {
      if (t >= 100) Console.WriteLine("Goz!");
      else Console.WriteLine("Viz!");
   }
   else Console.WriteLine("Jeg!");

   Console.ReadLine();
}
```

2.6. forráskód. A pontok távolságát kiszámító feladat megoldása

```
static void Main(string[] args)
   { Console.Write("Elso pont x kordinátája:");
      int x1 = Convert.ToInt32(Console.ReadLine());

      Console.Write("Elso pont y kordinátája:");
      int y1 = Convert.ToInt32(Console.ReadLine());

      Console.Write("Második pont x kordinátája:");
      int x2 = Convert.ToInt32(Console.ReadLine());

      Console.Write("Második pont y kordinátája:");
      int y2 = Convert.ToInt32(Console.ReadLine());

      double tavolsag =
      Math.Sqrt((x1 - x2) * (x1 - x2) + (y2 - y1) * (y2 - y1));

         Console.Write("Távolság: {0}", tavolsag);

         Console.ReadLine();
   }
```

2.7. forráskód. A pontok távolságát kiszámító feladat megoldása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace I_1_pelda
{
   class I_1_pelda
   {
      static void Main(string[] args)
      {
         int osztalyzat;
         Console.Write("Kérem az elért pontszámot: ");
         int pont=int.Parse(Console.ReadLine());

         if (pont >= 0 && pont < 50) osztalyzat = 1;
         else if (pont >= 50 && pont < 65) osztalyzat = 2;
         else if (pont >= 65 && pont < 80) osztalyzat = 3;
         else if (pont >= 80 && pont < 90) osztalyzat = 4;
         else if (pont >= 90 && pont <= 100) osztalyzat = 5;
         else osztalyzat = 0;

         if (osztalyzat > 0)
         Console.WriteLine("A kapott érdemjegy: {0}.", osztalyzat);
         else Console.WriteLine("Hibás az adat!");

         Console.ReadLine();
      }
   }
}
```

2.8. forráskód. Az átlagot számító feladat megoldása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class I_2_pelda
   {
      static void Main(string[] args)
      {
         Random rnd = new Random();
         int mag;
         int szorzo;
         int hozam;

         Console.Write ("Búza mennyisége tonnában: ");
         mag = int.Parse(Console.ReadLine());
         szorzo = rnd.Next(5,16);
         hozam = mag * szorzo;

         Console.WriteLine("A várható mennyiség {0}.", hozam);

         if (szorzo >= 5 && szorzo <= 8)
             Console.WriteLine("Átlag alatti év várható.");
         else if (szorzo >= 9 && szorzo <= 12)
             Console.WriteLine("Átlagos év várható.");
         else if (szorzo >= 13 && szorzo <= 15)
             Console.WriteLine("Átlag feletti év várható.");
         Console.ReadLine();
      }
   }
}
```

2.9. forráskód. A véletlen számokat generáló program forráskódja

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;

namespace feladat1_1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("Kerem az elso szamot: ");
         int szam1 = int.Parse(Console.ReadLine());
         Console.Write("Kerem a masodik szamot: ");
         int szam2 = int.Parse(Console.ReadLine());

         Random veletlen = new Random();

         Console.WriteLine("A generalt szamok: {0}, {1}, {2}.",
            veletlen.Next(szam1, szam2),
            veletlen.Next(szam1, szam2),
            veletlen.Next(szam1, szam2));

         Console.ReadLine();
      }
   }
}
```

2.10. forráskód. A dinnyecsomagoló feladat megoldása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace dinnyek
{
class labda
{
static void Main(string[] args)
   {
    Console.WriteLine("Dinnyek atmeroje(cm):!");
    int d = int.Parse(Console.ReadLine());
    Console.WriteLine();
    Console.WriteLine("Dinnyek szama!");
    int n = int.Parse(Console.ReadLine());
    double szalag = ((2 * d * Math.PI) + 60) * n;
    Console.WriteLine();
    Console.WriteLine("A szükséges szalag {0:0.00} cm.", szalag);
    Console.ReadLine();
   }
 }
}
```

2.11. forráskód. A burkolat mennyiségét kiszámító program forrása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
class csempe
{
static void Main(string[] args)
   {
    Console.Write("A szélesség méterben: ");
    double sz = double.Parse(Console.ReadLine());
    Console.WriteLine();
    Console.Write("A hosszúság méterben: ");
    double h = double.Parse(Console.ReadLine());
    double t = sz*h;
    Console.WriteLine();
    Console.WriteLine("A konyhánk területe: {0} m2", t);
    double cs = 0.2 * 0.2;
    double db = t/cs;
    double osszes=db+0.1*db;
    Console.WriteLine();
    Console.WriteLine
      ("A szüséges csempe mennyisége: {0:0.00} db", osszes);
    Console.ReadLine();
    }
}
}
```

2.12. forráskód. Másodfokú egyenletet kiszámító program forrása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace feladat_1
{
   class feladat_1
   {
      static void Main(string[] args)
      {
         Console.WriteLine
         ("Adja meg a másodfokú egyenlet együtthatóit!");
         Console.WriteLine();
         Console.Write("Kérem az a együttható értékét: ");
```

```
            double a = double.Parse(Console.ReadLine());
            Console.Write("Kérem a b együttható értékét: ");
            double b = double.Parse(Console.ReadLine());
            Console.Write("Kérem a c együttható értékét: ");
            double c = double.Parse(Console.ReadLine());
            double d = b * b - 4 * a * c;
            Console.WriteLine();
            if (d == 0)
             Console.WriteLine("Egy valós gyöke van az egyenletnek.");
            else if (d > 0)
             Console.WriteLine("Két valós gyöke van az egyenletnek.");
            else Console.WriteLine("Nincs valós gyöke az egyenletnek.");
            Console.ReadLine();
        }
    }
}
```

3. fejezet - Ujjgyakorlatok (szerző: Király Roland)

Tartalom

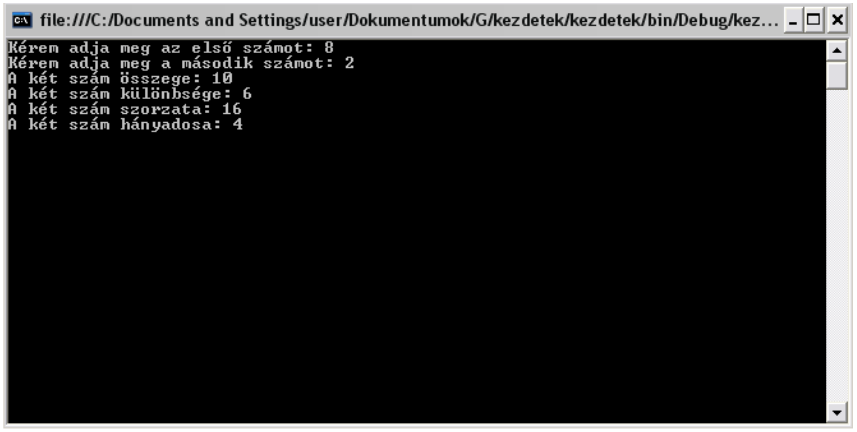
A fejezet forráskódjai

3.1. feladat (Alapvető műveletek – *szint: 1*). Készítsünk programot, mely bekér két számot, majd kiírja az összegüket, a különbségüket, a szorzatukat és a hányadosukat. Az adatokat a billentyűzetről olvassuk be. A beolvasást mindaddig végezzük, míg helyes adatokat nem kapunk.

Magyarázat: Az eredményeket nem kell tárolni, mivel a kiszámításukhoz szükséges kifejezést elhelyezhetjük a kiíró utasításban is. A C# nyelvben a *Write* és a *WriteLine* képes elvégezni a kifejezésben leírtakat, és az eredményüket megjeleníteni a képernyőn. Ezzel a megoldással tárterületet takaríthatunk meg.

A 3.1 forrásszövegben találjuk a megoldást, a program kimenetét pedig a 3.1 ábrán láthatjuk.

3.1. ábra. Az „alapvető műveletek” feladat megoldása



3.2. feladat (Kimenet formázása – *szint: 1*). Olvassunk be a billentyűzetről egy számot, majd írjuk ki a szám kétszeresét a képernyőre. A beolvasott számot és az eredményt nem kell mindenképpen tárolni.

Magyarázat: A beolvasott szám kétszeresének kiszámítását a kiírásban is elvégezhetjük. Ehhez használjuk a *Condole.WriteLine* metódust. A kiírás során a kimenetet formázhatjuk is az alábbi formulával:

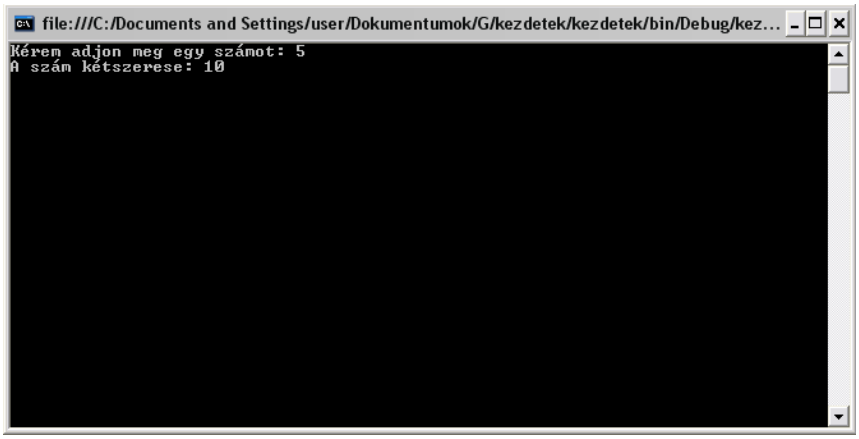
```
Console.WriteLine("{0} kétszerese = {1}", ...)
```

A formázott kiírásban a `{0}` azt jelenti, hogy a paraméter listában elhelyezett első elemet kell kiírni elsőként. A `{1}` jelentése hasonló, csak itt a második elemre hivatkozunk.

Figyelem! A beolvasásnál a *ReadLine* használata mellett szöveges formában kapjuk meg a számot, ezért azt konvertálnunk kell számmá.

A 3.2 forrásszövegben találjuk a megoldást, a program kimenetét pedig a 3.2 ábrán láthatjuk.

3.2. ábra. Az „kimenet formázása” feladat megoldása

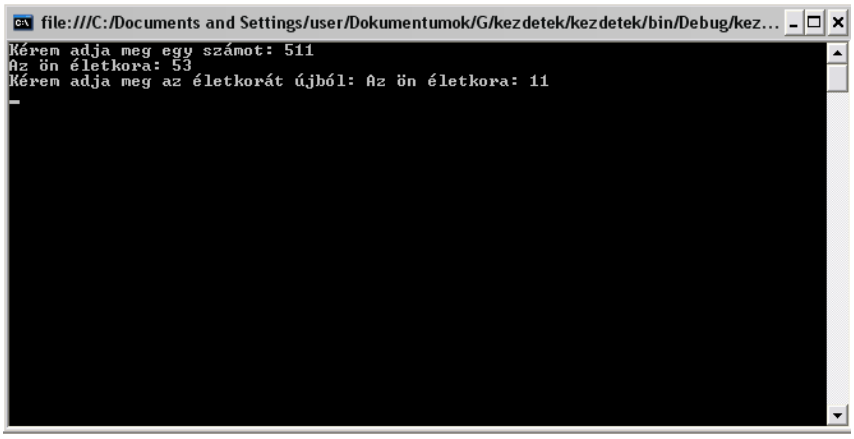


3.3. feladat (Read vagy ReadLine – *szint: 1*). Egy egyszerű program segítségével vizsgáljuk meg, hogy mi a különbség a `Console.Read` és a `Console.ReadLine` működése között.

Magyarázat: A Read és a ReadLine alapvetően a beolvasott adat típusában különböznek egymástól. Míg az első szöveges adatot olvas be, a második a megadott karakter kódjával tér vissza, vagyis egy számmal. Ezért, ha számokat olvasunk be, akkor a *ReadLine*-t, amennyiben csak egy karaktert, a *Read*-et kell használnunk.

A 3.3 forrásszövegben találjuk a megoldást, a program kimenetét pedig a 3.3 ábrán láthatjuk.

3.3. ábra. A „Read vagy ReadLine” feladat megoldása



3.4. feladat (Konzol képernyő használata – *szint: 1*).

Írassuk ki a képernyőre az alábbi számsorozatokat:

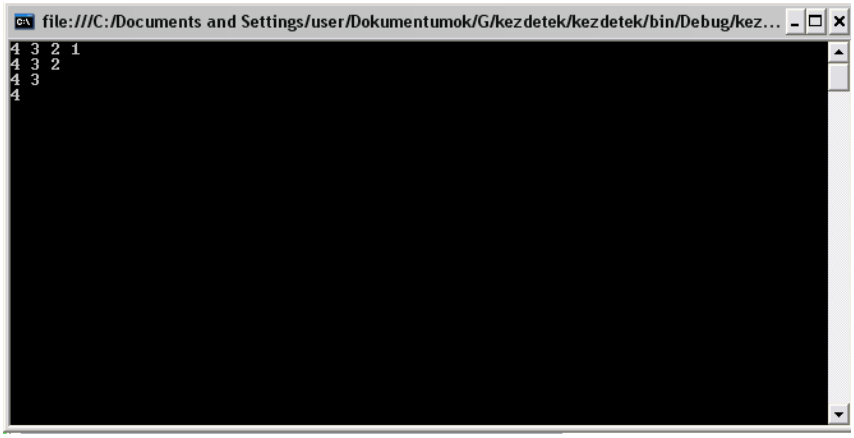
```
4 3 2 1
4 3 2
4 3
4
```

Próbáljuk meg úgy elkészíteni a programot, hogy a `Console.WritemelletLine()` csak egyszer szerepeljen a programban.

Magyarázat: A kiíró utasításban a formázáshoz elhelyezhetünk a kiírandó szövegben `\n` jeleket, melyek törik a sort. Ezzel megoldható a lehető legkevesebb kiírás használata mellett a kívánt kimenet előállítása.

A 3.4 forrásszövegben találjuk a megoldást, a program kimenetét pedig a 3.4 ábrán láthatjuk.

3.4. ábra. Az „Konzol képernyő használata” feladat megoldása



3.5. feladat (Szállásszervezés – *szint: 1*). Készítsünk programot, mely osztálykirándulás szervezésében segíti a használóját, melyhez a lehető legkedvezőbb szállásárat kellene elérni. A kiválasztott hotelben többféle kedvezményt adnak a diákoknak, egyszerre közülük csak az egyik vehető igénybe:

- Csoportos kedvezmény: 10 fő alatt 0 %; 10-19 fő esetén 5 %; 20-29 fő esetén 8 %; 30-40 fő esetén 12 %; 40 fő felett 14 % a kedvezmény mértéke.
- Intézményi kedvezmény: 5 fő alatt nincs; 5-11 fő esetén 1 fő ingyen szálláshoz jut; 12-19 fő esetén 2 fő ingyenes; 20-28 fő esetén 3 fő ingyenes; 29-40 fő esetén 4 fő, míg 40 fő felett 5 fő kap ingyenes szállást.
- Diákkedvezmény: egyénileg is jár, mértéke 10

Készítsen programot, amely beolvassa a kiránduláson résztvevők számát majd megadja, hogy a háromféle kedvezményből melyiket kell igénybe venni, hogy a lehető legkevesebbe kerüljön a szállás!

3.6. feladat (Kocka – *szint: 2*). Egy n cm (n>1 egész szám) oldalhosszúságú fakockát piros festékbe mártunk, majd 1 cm élű kiskockákra felfűrészeljük. Hány kis kocka lesz, amelynek

- pontosan egy oldallapja pirosra festett?
- pontosan két oldallapja piros?
- pontosan 3 lapja piros?

- egyik lapja sem piros?

Készítsünk C# programot, amely a felvázolt problémát implementálja!

3.7. feladat (Melyik szám a nagyobb – *szint: 1*). Készítsünk konzolos alkalmazást, amely bekér két egész számot, majd eldönti, hogy melyik a nagyobb. A két számot *int* típusú változókban tároljuk el. Amennyiben a két megadott szám azonos értékű, a bekérést ismételjük meg.

Magyarázat: A megoldáshoz használjunk feltételes elágazást. Rossz adatok megadásakor az ismétlést folytassuk mindaddig, amíg helyes adatokat nem kapunk.

A 3.5 forrásszövegben találjuk meg a feladat megoldását.

3.8. feladat (Osztályzatok – *szint: 1*). Írjon programot, amely bekér egy informatika osztályzatot, majd kiírja a szülők véleményét az eredményről. A program a „nemlétező” osztályzatokra is reagáljon.

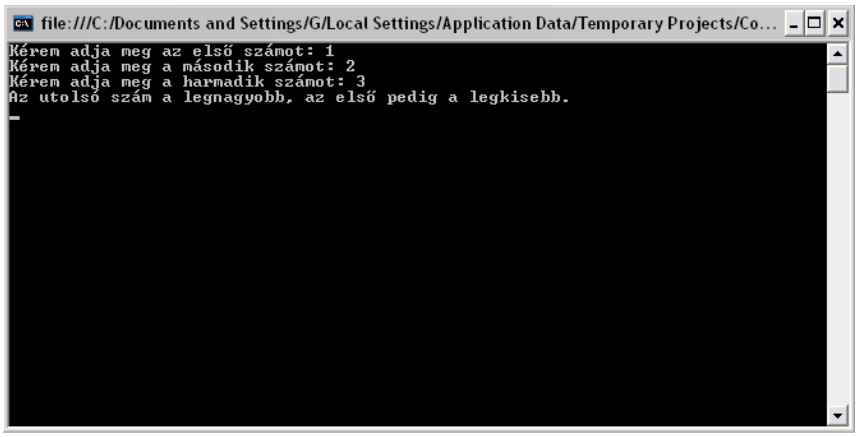
A 3.7 forrásszövegben találjuk a megoldást.

3.9. feladat (Számok sorrendje – *szint: 1*). Kérjünk be a billentyűzetről három egész számot, majd döntsük el, hogy melyik a legnagyobb, és a legkisebb érték.

Magyarázat: Ez a feladat hasonlít az ismert rendező algoritmusokra annyiban, hogy a kapott értékeket sorrendbe rakja, de a megoldás nem rugalmas, mivel a rendezendő elemek száma erősen korlátozott…

A 3.6 forrásszövegben találjuk a megoldást, a program kimenetét pedig a 3.5 ábrán láthatjuk.

3.5. ábra. Az „Konzol képernyő használata” feladat megoldása

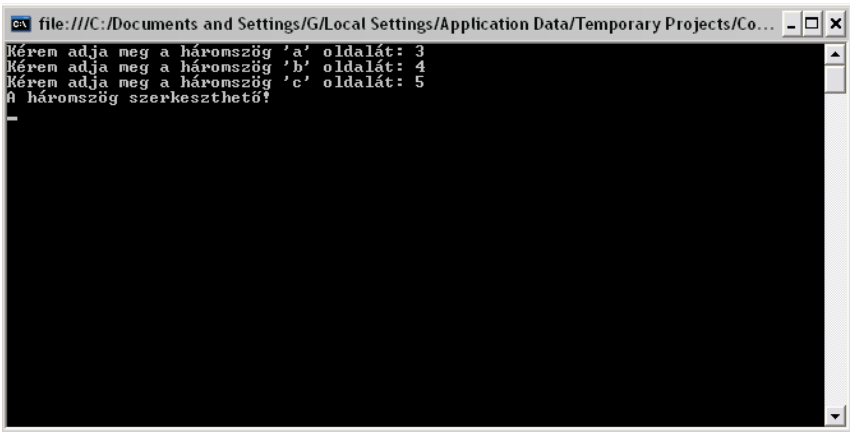


3.10. feladat (Szerkeszthető háromszögek – *szint: 1*). Készítsünk konzol programot, amely bekér három egész számot a billentyűzetről. A bekért számokra úgy tekintünk, mint egy háromszög oldalaira. Döntsük el, hogy a háromszög szerkeszthető-e.

Magyarázat: A háromszög abban az esetben szerkeszthető, ha bármely két oldal hosszának az összege nagyobb a harmadik oldal hosszánál.

A 3.8 forrásszövegben találjuk a megoldást, a program kimenetét pedig a 3.6 ábrán láthatjuk.

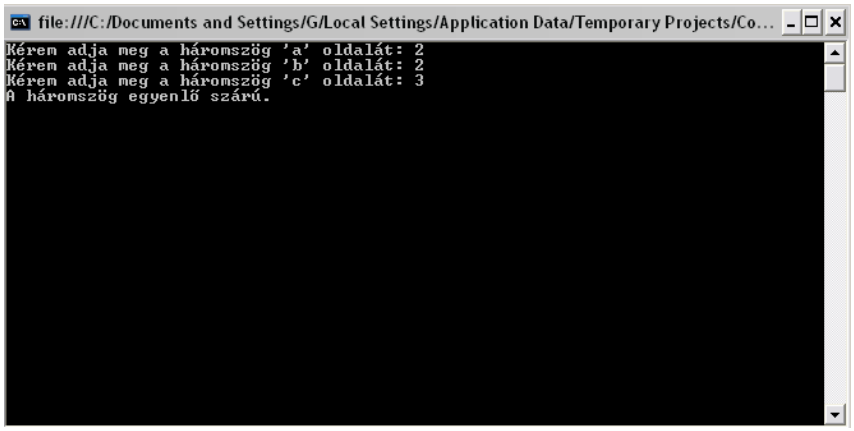
3.6. ábra. Az „Szerkeszthető háromszögek” feladat megoldása



3.11. feladat (Háromszög típusa – *szint: 1*). Készítsünk konzol programot, amely bekér három egész számot a billentyűzetről. A bekért számokra úgy tekintünk, mint egy háromszög oldalaira. Döntsük el, hogy a háromszög egyenlő oldalú, illetve egyenlő szárú háromszög-e.

A 3.9 forrásszövegben találjuk a megoldást, a program kimenetét pedig a 3.7 ábrán láthatjuk.

3.7. ábra. Az „Háromszög típusa” feladat megoldása

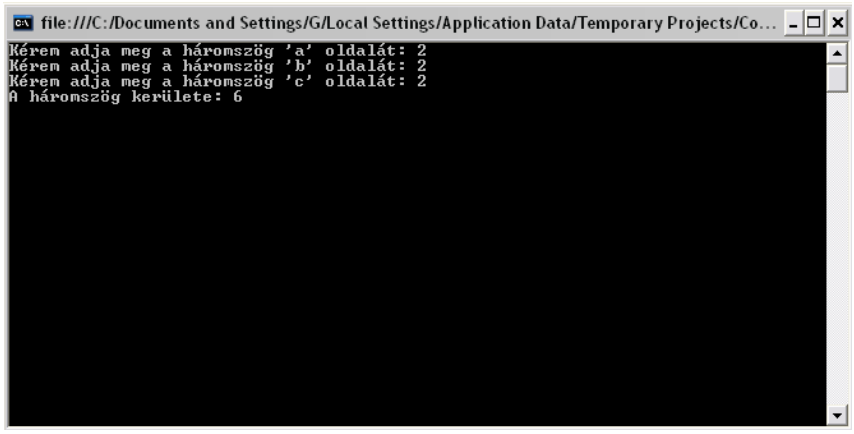


3.12. feladat (Háromszög kerülete – *szint: 1*). Készítsünk konzol programot, amely bekér három egész számot a billentyűzetről. A bekért számokra úgy tekintünk, mint egy háromszög oldalaira. Számítsuk ki a háromszög kerületét és területét.

Magyarázat: A kerület kiszámítása nem okoz különösebb problémát, mivel egyenlő az oldalak hosszának az összegével. Amennyiben helyes programot szeretnénk készíteni figyeljünk arra is, hogy a háromszög szerkeszthető-e.

A 3.10 forrásszövegben találjuk a megoldást, a program kimenetét pedig a 3.8 ábrán láthatjuk.

3.8. ábra. Az „alapvető műveletek” feladat megoldása



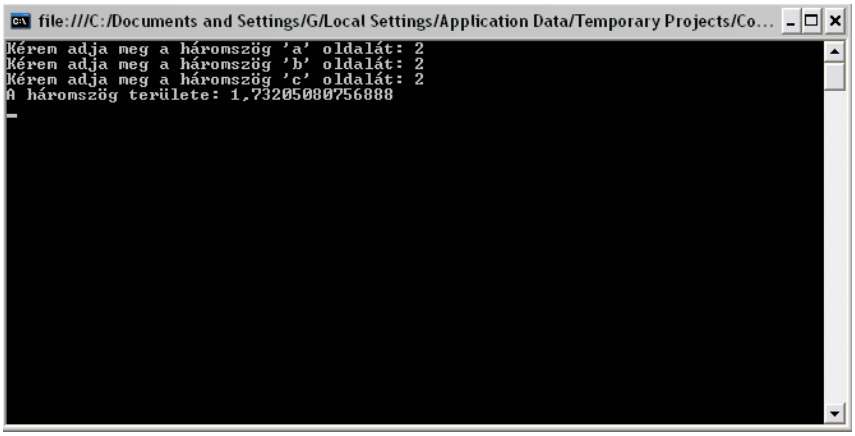
3.13. feladat (Háromszög területe - Héron képlet – *szint: 2*). Készítsünk konzol programot, amely bekér három egész számot a billentyűzetről. A bekért számokra úgy tekintünk, mint egy háromszög oldalaira. Számítsuk ki a háromszög területét. A terület kiszámításához használhatjuk a Hérón képletet.

Magyarázat: A Hérón képlet segítségével a háromszög területét az oldalak hosszából is ki tudjuk számolni $T = \sqrt{(s(s-a)(s-b)(s-c))}$

Az a, b és c a háromszög oldalai a $s = \frac{a+b+c}{2}$ képlettel számolhatók ki, ahol S a háromszög kerületének a fele

A 3.11 forrásszövegben találjuk a megoldást, a program kimenetét pedig a 3.13 ábrán láthatjuk.

3.9. ábra. A háromszög területe



3.14. feladat (Majdnem Lottó – *szint: 1*). Generáljunk tíz darab 1-6 közé eső véletlen számot. A program ezután mondja meg hányszor volt hatos a generált érték!

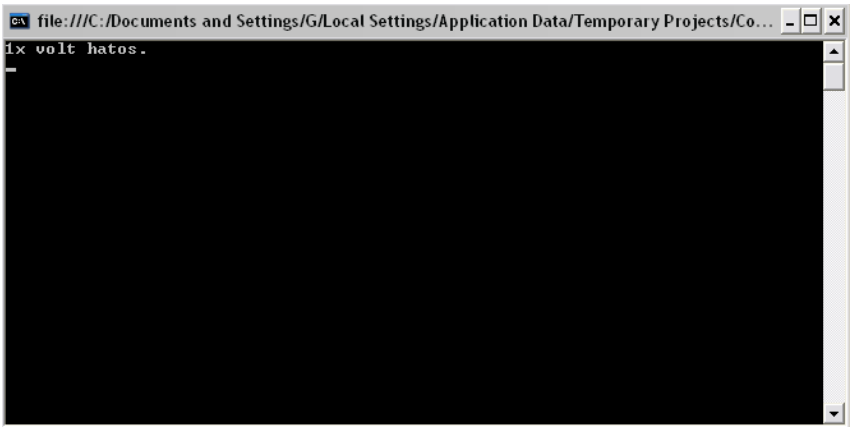
Magyarázat: A véletlen számok generálásához használjuk a *Random* osztály szolgáltatásait.

```
  Random R = new Random();
int adat = R.Next();
```

A generált számokat nem kell tárolnunk, mivel minden értékről azonnal eldönthető, hogy az 6-os, vagy sem.

A 3.12 forrásszövegben találjuk a megoldást, a program kimenetét pedig a 3.10 ábrán láthatjuk.

3.10. ábra. A majdnem Lotto program kimenete



3.15. feladat (Szóközök – *szint: 1*). Kérjünk be egy mondatot, majd írjuk ki szóközök nélkül. A 3.13 forrásszövegben találjuk a megoldást.

3.16. feladat (Sorozatok – *szint: 2*). Készítsünk olyan konzolos alkalmazást, amely beolvassa egy számtani sorozat első elemét, valamint a differenciáját, és egy tetszőleges *N* értéket, majd kiírja a sorozat n. elemét, és az első *N* tagja összegét.

3.17. feladat (Command Line Interface – *szint: 1*). Készítsünk egy egyszerű parancssori programot, amely néhány menüponttal rendelkezik. A menüpontok kiírására használjuk a *Console* osztály kiíró utasításait.

A menü a következőképp nézzen ki:

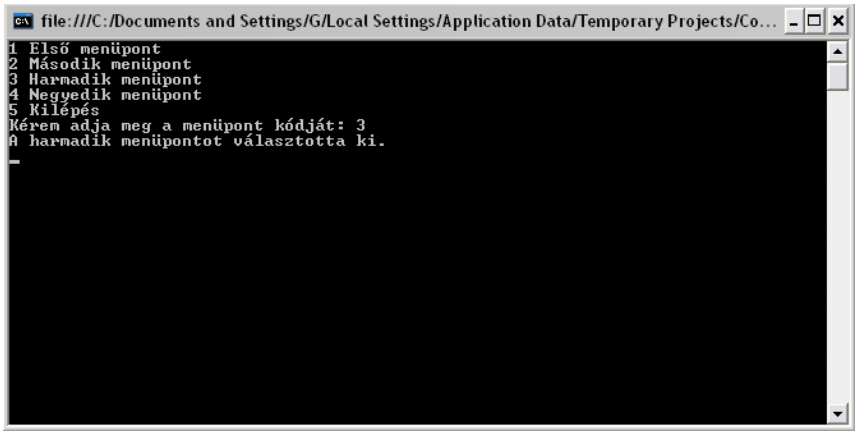
```
1 Első menüpont
2 második menüpont
3 Harmadik menüpont
4 Negyedik menüpont
5 Kilépés
```

A program közvetlenül az elindítása után írja ki a menüket a képernyőre, majd olvasson be egy karaktert. Amennyiben a beolvasott adat az 1-5 intervallumba eső szám, úgy a képernyőre íródjon ki, hogy melyik menüpont került kiválasztásra, ellenkező esetben jelenjen meg a *Rossz választás felirat*.

Magyarázat: A program elkészítése során alkalmazhatjuk a *switch* vezérlő szerkezetet annak az eldöntésére, hogy a beolvasott szám beleesik-e a menüpontoknál definiált intervallumba. Hiba esetén a *switch default* ága írja ki a hibaüzenetet a képernyőre.

A 3.14 forrásszövegben találjuk a megoldást, a program kimenetét pedig a 3.11 ábrán láthatjuk.

3.11. ábra. A majdnem Lottó program kimenete

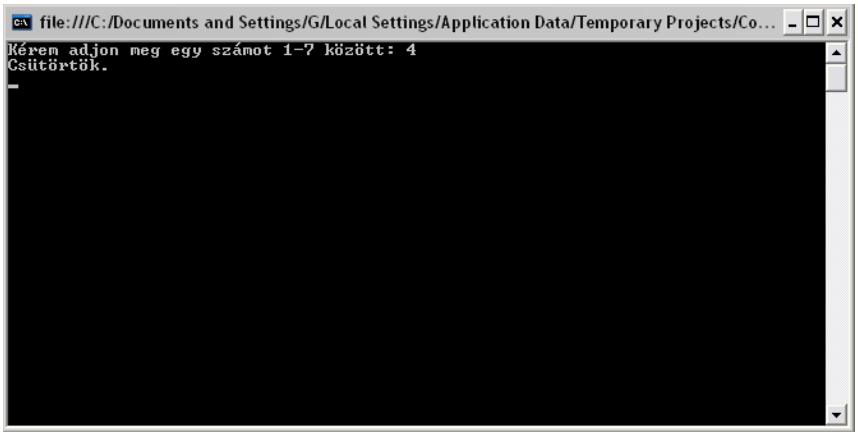


3.18. feladat (A hét napjai – *szint: 1*). Készítsünk konzolos alkalmazást, amely paraméterként kap egy egész számot (int), majd kiírja a hét azonos sorszámú napját a képernyőre.

Az 1-es érték jelenti a hétfőt, a 2-es a keddet, a 7-es a vasárnapot. Amennyiben a megadott szám nem esik az 1-7 intervallumba, a program írjon hibaüzenetet a képernyőre.

A 3.15 forrásszövegben találjuk a megoldást, a program kimenetét pedig a 3.12 ábrán láthatjuk.

3.12. ábra. A majdnem Lottó program kimenete



3.19. feladat (Életkorok – *szint: 1*). Készítsünk alkalmazást, amely beolvassa egy személy életkorát (E), majd a kapott adat fényében kiírja a képernyőre azt a korosztályt, amibe az életkor „tulajdonosa” tartozik.

- Gyermek (0-6),
- Iskolás (7-22),
- Felnőtt (22-64),
- 65 töl nyugdíjas!

A 3.16 forrásszövegben találjuk a megoldást, a program kimenetét pedig a 3.13 ábrán láthatjuk.

3.13. ábra. Az életkoros feladat kimenete

A fejezet forráskódjai

3.1. forráskód. Alapvető műveleteket megvalósító program

```
using System;
using System.Collections.Generic;
using System.Text;

namespace kezdetek
{
 class Program
 {
    static void Main(string[] args)
    {
      Console.Write
           ("Kérem adja meg az első számot: ");
      double a = double.Parse(Console.ReadLine());
      Console.Write
           ("Kérem adja meg a második számot: ");
      double b = double.Parse(Console.ReadLine());

      Console.WriteLine
           ("A két szám összege: {0}", a + b);
      Console.WriteLine
           ("A két szám különbsége: {0}", a - b);
      Console.WriteLine
           ("A két szám szorzata: {0}", a * b);
      Console.WriteLine
           ("A két szám hányadosa: {0}", a / b);
      Console.ReadLine();
    }
  }
}
```

3.2. forráskód. Kimenet formázását végző program forrása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace kezdetek
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write
         ("Kérem adjon meg egy számot: ");
         Console.WriteLine
           ("A szám kétszerese: {0}",
           (int.Parse(Console.ReadLine()))*2);
         Console.ReadLine();
      }
   }
}
```

3.3. forráskód. Read és ReadLine

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace kezdetek
{
   class Program
   {
      static void Main(string[] args)
      {
        Console.Write
          ("Kérem adja meg egy számot: ");
        int a = Console.Read();
        Console.WriteLine
          ("Az ön életkora: {0}", a);

        Console.Write
          ("Kérem adja meg az életkorát újból: ");
        int b = int.Parse(Console.ReadLine());
        Console.WriteLine
          ("Az ön életkora: {0}", b);

        Console.ReadLine();
      }
   }
}
```

3.4. forráskód. A konzol képernyő használatát bemutató program forrása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace kezdetek
{
   class Program
   {
      static void Main(string[] args)
      {
       Console.Write("4 3 2 1\n4 3 2\n4 3\n4");
       Console.ReadLine();
      }
   }
}
```

3.5. forráskód. Melyik szám a nagyobb

```
namespace kezdetek
{
 class Program
 {
   static void Main(string[] args)
    {
       Console.Write ("Első szám: ");
       int elso = int.Parse(Console.ReadLine());
       Console.Write ("Az előzőtől külömböző szám: ");
       int masodik =
                int.Parse(Console.ReadLine());
       if (elso == masodik)
       {
         while (elso == masodik)
         {
         Console.Write
         ("Hiba, ... ismét : ");
          masodik = int.Parse(Console.ReadLine());
         }
       }
       if (elso > masodik)
       Console.WriteLine ("Az első szám a nagyobb!");
       if (elso < masodik)
       Console.WriteLine ("A második szám a nagyobb!");
       Console.ReadLine();
```

```
      }
   }
}
```

3.6. forráskód. A Számok sorrendje feladat

```
namespace sorrend
{
 class Program
 {
  static void Main(string[] args)
  {
     Console.Write ("Első szám: ");
     int a = int.Parse(Console.ReadLine());
     Console.Write ("Második szám: ");
     int b = int.Parse(Console.ReadLine());
     Console.Write ("Harmadik szám: ");
     int c = int.Parse(Console.ReadLine());
     if (a > b && a > c && b > c){
       Console.WriteLine("Az első szám a legnagyobb," +
                    "az utolsó pedig a legkisebb.");
     }
     if (a > b && a > c && b < c){
       Console.WriteLine ("Az első szám a legnagyobb," +
                    "a középső pedig a legkisebb.");}
     if (a < b && a > c && b > c){
      Console.WriteLine
          ("A középső szám a legnagyobb, az utolsó " +
           "pedig a legkisebb.");
     }
     ...
     Console.ReadLine();
  }
}
}
```

3.7. forráskód. Az Osztályzatok feladat megoldása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace jegyek
{
   class osztalyzat
   {
      static void Main(string[] args)
      {
         Console.Write("A dolgozatod eredménye (számmal): ");
         string osztalyzat = Console.ReadLine();
         Console.WriteLine("\nSzüleid véleménye:\n");
         switch (osztalyzat)
         {
         case "1":
               Console.WriteLine
               ("Megmondtam, hogy ez lesz a vége,"+
                "ha csak játékra használod a számítógépet!!!");
               Console.WriteLine
               ("Büntetés: Egy hétig nincs se Tv, se Internet! ");
               break;
            case "2":
               Console.WriteLine
                  ("Megmondtam, hogy olvasd még át legalább"+
                  " egyszer lefekvés előtt!!!");
               Console.WriteLine
                  ("Büntetés: Ma este nincs se Tv, se"+
                  "Internet! Alvás, és kész.");
               break;
            case "3":
               Console.WriteLine
                ("Ha egy kicsit többet gyakorolnál,"+
                 "akkor még jobb is lehetne!");
               break;
            case "4":
               Console.WriteLine
                    ("Szép - szép, de ugye évvégére" +
                     "kijavítod ötösre?!");
               break;
         ...
         }
         Console.ReadLine();
      }
   }
}
```

3.8. forráskód. Az első háromszöges feladat megoldása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
```

```
         Console.Write ("Háromszög 'a' oldala: ");
         int a = int.Parse(Console.ReadLine());
         Console.Write ("Háromszög 'b' oldala: ");
         int b = int.Parse(Console.ReadLine());
         Console.Write ("Háromszög 'c' oldala: ");
         int c = int.Parse(Console.ReadLine());
         if (a+b>c&&a+c>b&&b+c>a)
         {
          Console.WriteLine ("A háromszög szerkeszthető!");
         }
         else {
          Console.WriteLine ("A háromszög nem szerkeszthető!");
         }
         Console.ReadLine();
      }
   }
}
```

3.9. forráskód. A Háromszög típusa című feladat megoldása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write ("Háromszög 'a' oldala: ");
         int a = int.Parse(Console.ReadLine());
         Console.Write("Háromszög 'b' oldala: ");
         int b = int.Parse(Console.ReadLine());
         Console.Write("Háromszög 'c' oldala: ");
         int c = int.Parse(Console.ReadLine());
         if (a == b && a == c && c == b){
          Console.WriteLine("Egyenlő oldalú.");
         }
         else if (a == b || a == c || c == b)
          Console.WriteLine("Egyenlő szárú.");
         else Console.WriteLine("Nem egyenlő oldalú" +
                         "és nem is egyenlő szárú.");
         Console.ReadLine();
      }
   }
}
```

3.10. forráskód. A Háromszög kerülete feladat megoldása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write ("Háromszög 'a' oldala: ");
         int a = int.Parse(Console.ReadLine());
         Console.Write("Háromszög 'b' oldala: ");
         int b = int.Parse(Console.ReadLine());
         Console.Write ("Háromszög 'c' oldala: ");
         int c = int.Parse(Console.ReadLine());
         Console.WriteLine
           ("A háromszög kerülete: {0}", a+b+c);
         Console.ReadLine();
      }
   }
}
```

3.11. forráskód. A Háromszög területe feladat megoldása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write ("Háromszög 'a' oldala: ");
         double a = int.Parse(Console.ReadLine());
         Console.Write ("Háromszög 'b' oldala: ");
         double b = int.Parse(Console.ReadLine());
         Console.Write ("Háromszög 'c' oldala: ");
         double c = int.Parse(Console.ReadLine());
         double s = (a + b + c) / 2;
         Console.WriteLine ("A háromszög kerülete: {0}",
```

```
                        Math.Sqrt(s*(s-a)*(s-b)*(s-c)));
            Console.ReadLine();
        }
    }
}
```

3.12. forráskód. A Háromszög területe feladat megoldása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Random rnd = new Random();
         int hanyszor = 0;
         for (int i = 0; i < 10; i++)
         {
            if (rnd.Next(1, 7) == 6)
               hanyszor = hanyszor + 1;
         }
         Console.WriteLine
            ("{0}x volt hatos.", hanyszor);
         Console.ReadLine();
      }
   }
}
```

3.13. forráskód. A szóközös feladat megoldása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace feladat_5
{
   class feladat_5
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Kérem gépeljen be egy mondatot!");
         string mondat = Console.ReadLine();
         for (int i = 0; i < mondat.Length; i++)
         {
            if (mondat[i]!=' ')
            {
               Console.Write(mondat[i]);
            }
         }
         Console.ReadLine();
      }
   }
}
```

3.14. forráskód. Command Line Interface

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("1 Első menüpont");
         Console.WriteLine("2 Második menüpont");
         Console.WriteLine("3 Harmadik menüpont");
         Console.WriteLine("4 Negyedik menüpont");
         Console.WriteLine("5 Kilépés");
         Console.Write("Menüpont kódja: ");
         int melyik=int.Parse(Console.ReadLine());
         switch(melyik)
         {
            case 1:
               Console.WriteLine
               ("Az első menüpontot választotta.");
               break;
            case 2:
               Console.WriteLine
               ("A második menüpontot választotta ki.");
               break;
            case 3:
               Console.WriteLine
               ("A harmadik menüpontot választotta ki.");
               break;
            case 4:
               Console.WriteLine
               ("A negyedik menüpontot választotta ki.");
               break;
            case 5:
               Console.WriteLine
```

```
                ("A kilépés menüpontot választotta ki.");
                break;
            }
            Console.ReadLine();
        }
```

3.15. forráskód. A hét napjai

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write ("Szám 1-7 között: ");
         int napkod=int.Parse(Console.ReadLine());
         switch(napkod){
            case 1:Console.WriteLine("Hétfő."); break;
            case 2:Console.WriteLine("Kedd.");   break;
            case 3:Console.WriteLine("Szerda.");  break;
            case 4:Console.WriteLine("Csütörtök.");break;
            case 5:Console.WriteLine("Péntek."); break;
            case 6:Console.WriteLine("Szombat."); break;
            case 7:Console.WriteLine("Vasárnap."); break;
            default:Console.WriteLine ("Rossz kódot adott meg.");
            break;
         }
         Console.ReadLine();
      }
   }
}
```

3.16. forráskód. Az életkorokat vizsgáló feladat megoldása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {

         Console.Write
           ("Kérem adja meg az életkorát: ");
         int E =
            int.Parse(Console.ReadLine());
         int a=0;
         if (E >= 0 && eletkor < 7) a=1;
         if (E >= 7 && eletkor < 22) a = 2;
         if (E >= 19 && eletkor < 66) a = 3;
         if (E > 65) a = 4;
         switch (a)
         {
            case 1:
               Console.WriteLine("Gyermek.");
               break;
            case 2:
               Console.WriteLine("Iskolás.");
               break;
            case 3:
               Console.WriteLine("Felnőtt.");
               break;
            case 4:
               Console.WriteLine("Nyugdíjas.");
               break;
            default:
               Console.WriteLine
                 ("Rossz értéket adott meg.");
               break;
         }
         Console.ReadLine();
      }
   }
}
```

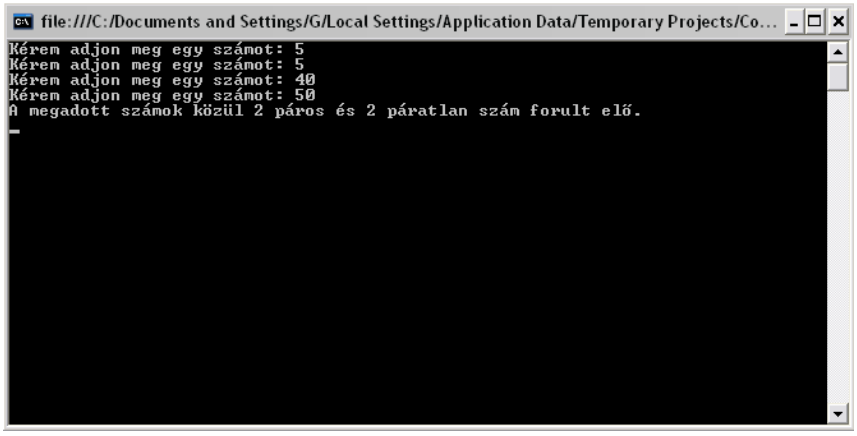
4. fejezet - Ciklusokhoz kapcsolódó feladatok (szerző: Király Roland)

Tartalom

4.1. feladat (Több elem bekérése – *szint: 1*). Írjunk olyan programot, amely addig kér be egész számokat a billentyűzetről, amíg azok összege meg nem haladja a 100-at. A beolvasás végén írjuk ki azt, hogy a bekért számok közül hány volt páros, és hány volt páratlan.

A 4.1 forrásszövegben találjuk a megoldást, a program kimenetét pedig a 4.1 ábrán láthatjuk.

4.1. ábra. A tömb bekérése



4.2. feladat (Mátrix bekérése – *szint: 2*). Kérjük be egy 2x2-es (esetleg egy 3x3-as) mátrix elemeit, majd „rajzoljuk” ki a mátrixot a konzol képernyőre, végül számítsuk ki és írjuk ki a determinánsát.

Magyarázat: Determinánson egy négyzetes mátrixhoz rendelt számot értünk. Ha egy A nxn-es négyzetes mátrix elemei az $a[i,j]$ számok, akkor az (n-ed rendű) determináns a Leibniz-formula segítségével kapható meg.

A 4.2 forrásszövegben találjuk a megoldást, a számítás menetét az alábbiakban láthatjuk.

$$\begin{vmatrix} 2 & 5 \\ 3 & 1 \end{vmatrix} = 2 \cdot 1 - 5 \cdot 3 = -13$$

4.3. feladat (Négyszögek – *szint: 1*). Gyakoroljunk a konzollal. Készítsünk programot, amely képernyő közepére a bal széltől a jobb szélig tartó négyszöget rajzol. A 4.3 forrásszövegben találjuk a megoldást.

4.4. feladat (Számláló és nevező – *szint: 1*). Készítsünk programot, mely egy tört számlálójának és nevezőjének megadása után kiírja az egyszerűsített törtet. A 4.4 forrásszövegben találjuk meg a feladat megoldását.

4.5. feladat (Egerek – *szint: 3*). Tételezzük fel, hogy létezik teljesen szeparált, *L* egység hosszúságú csatorna, és mindkét végénél egy-egy egér. Egy indító jelre az egyik egér *U*, a másik *V* sebességgel kezd rohanni a csatorna ellenkező vége felé. Amikor odaérnek, visszafordulnak és újra egymással szemben haladnak (faltól falig rohangálnak). Három módon találkozhatnak, néha szemből, néha a gyorsabb utoléri a lassabbat, néha pedig egyszerre érnek egy falhoz…

Készítsünk olyan programot, ami bekéri egy képzeletbeli szennyvízcsatorna hosszát (L), két patkány sebességét (U és V), valamint egy időtartamot (T), majd kiírja, hogy a megadott időtartam alatt a patkányok hányszor találkoztak.

4.6. feladat (Karakterek bekérése – *szint: 1*). Írjunk programot, mely bekér 5 különböző karaktert, majd kiírja ezen karakterek összes permutációját. A program egy lehetséges megoldását a 4.6 forrásszövegben találjuk.

4.7. feladat (Összegek kiszámítása – *szint: 1*). Készítsünk alkalmazást a 4.7 forrásszövegben látható program mintájára, amely bekéri a K pozitív egész számot, majd kiszámolja a következő összeget: $1*2+2*3+3*4+4*5+\cdots+K*(K+1)$

4.8. feladat (Háromszög kirajzolása – *szint: 1*). A 4.8 programszöveg mintájára kérjünk be egy természetes számot (a), majd rajzoljunk ki a képernyőre egy derékszögű háromszöget csillagokból (*). A háromszög pontosan az a-val megegyező sornyi csillagból álljon.

4.9. feladat (Betűk rajzolása a képernyőre – *szint: 1*). Készítsünk programot, amely bekér egy N természetes számot, majd kirajzol a képernyőre egymás mellé N-szer az "XO" betűket. A feladat egy lehetséges megoldását a 4.9 programban találjuk meg.

4.10. feladat (Unalmas az informatika óra – *szint: 1*). Írjon programot, ami megkérdezi a felhasználótól, hogy hány másodperc „zenét” szeretne hallgatni. A megadott másodpercen keresztül szólaltassunk meg véletlen frekvenciájú hangokat, véletlen(1 és 600 ms között) ideig.

4.11. feladat (Betűkígyó – *szint: 2*). Oldja meg a 4.11 forrásszövegben látható program mintájára a következő feladatot. Piros színű konzol ablakban a képernyő bal szélétől kezdve kirajzolunk egy betűkígyót, ami folyamatosan növekszik (A-K-ig). Ha a billentyűzeten lenyomunk egy betűt, akkor a választottal megegyező betűk kikerülnek a kígyóból...

4.12. feladat (Tízes számrendszer – *szint: 2*). Készítsünk konzol programot, amely bekér (esetleg véletlenszerűen generál) egy bitsorozatot (2-es számrendszerbeli számot), majd átváltja 10-es számrendszerbebe. A feladat egyszerű megoldását megtaláljuk a 4.12 forrásszövegben.

Magyarázat: A véletlen számok generálásához használjuk a *Random* osztályt úgy, hogy paraméterezzük a *Next* metódusát. A paraméterezésre azért van szükség, hogy csak 0-1 számjegyeket generáljon.

4.13. feladat (C# logó – *szint: 1*). A 4.13 program mintájára rajzoljuk ki a képernyőre valamely általunk választott karakter felhasználásával a C# nyelv logóját.

Magyarázat: A megoldáshoz a karakteres képernyőn jól kell tudni pozicionálni, és érdemes a rendelkezésre álló helyet arányosan elosztani, hogy a logo megfelelően nézzen ki. A pozíciók megadásához használjuk a *Console.SetCursorPosition* metódust.

4.14. feladat (Csoportosítási feladat – *szint: 2*). A 4.14 főprogram mintájára készítsünk konzolos alkalmazást, amely beolvassa egy adott osztály névsorát. Alkossunk az osztályból véletlenszerűen csoportokat. A program kérdezze meg a használóját, hogy hány fős csoportokat szeretne létrehozni, majd írja ki azokat a képernyőre.

Magyarázat: Ahhoz, hogy csoportokat tudjunk készíteni, szükség lesz a tanulók neveinek és sorszámainak a tárolására. A programnak létezik egy kevésbé bonyolult megoldása is, ahol a csoportokban a nevek helyett az egyes tanulókat a sorszámuk jelöli. ennél a megoldásnál a beolvasást is lerövidíthetjük.

4.15. feladat (Monogramok – *szint: 1*). Kérjünk be egy nevet, majd írjuk ki a névhez tartozó monogramot.

Magyarázat: A 4.15 szövegben látható feladat megoldásánál problémába ütközhetünk, ha az adott személy kereszt, vagy családi neve kettős vagy több jelből álló

betűket tartalmaz. Ilyenek betűk a *CS, DZ, SZ, ZS, ...* A probléma megoldásához, vagyis a nem egy karakterből álló betűk kiszűrésére építsünk elágazást a programba (*switch*).

4.16. feladat (Számjegyek összege – *szint: 1*). Készítsünk a 4.16 program mintájára konzolos alkalmazást, amely beolvas egy számot, majd kiírja a számjegyek összegét a képernyőre.

Magyarázat: A beolvasott számot ne konvertáljuk számmá, ahogy azt a számításokat végző programok esetén tenni szoktuk, hanem hagyjuk meg *string* formában, mivel így könnyedén szét tudjuk szedni elemeire, vagyis a számjegyeire, melyeket azután át tudunk konvertálni számmá az összeadáshoz. Az konverzióhoz használjuk a *int.Parse()* metódust az adott *string* elemre (`szam[i].ToString()`). Figyelem! A *string* elemeit is konvertálnunk kell, mivel azok karakter típusúak.

4.17. feladat (Pénzérme feldobása – *szint: 1*). Készítsünk a 4.17 feladathoz hasonló programot, amely egy képzeletbeli pénzérmét dobál a levegőbe, majd megállapítja, hogy az a fej, vagy az írás oldalára esett-e le. A pénzérme feldobását követően a program írja ki a képernyőre az eredményt, vagyis, hogy hányszor volt fej, és hányszor írás. Természetesen a pénzérme dobások számát is a felhasználó adja meg a program elején.

4.18. feladat (Kukac – *szint: 3*). Készítsünk a 4.18 forrásszöveg alapján programot, melyben egy kukacot (@) a kurzor mozgató billentyűk segítségével mozgathatuk a képernyőn addig, amíg az ESC billentyűvel ki nem lépünk a programból.

Magyarázat: Amennyiben jól használható programot szeretnénk készíteni, figyelhetünk a képernyő szélének az elérésére, vagyis arra, hogy a kukac ne tudjon kilépni a megadott koordináták közül.

A legjobb megoldás az, ha a kukac eleje a szélek elérésekor az ellentétes oldalon bukkan ki, a háta pedig követi. Ezzel a megoldással egy un.: két dimenziós Missner-teret hozunk létre amely ugyanilyen alapokon nyugszik. Nagyon leegyszerűsítve, és kissé elbagatelizálva a dolgot úgy is mondhatnánk, hogy „elől ki, hátul be” típusú teret készítettünk…

4.19. feladat (Napszakok és órák – *szint: 1*). Készítsünk a 4.19 példához hasonló programot, amely az elindítása után a napszaknak megfelelően köszön!

4.20. feladat (Kamatszámítás – *szint: 3*). Írjunk konzolos alkalmazást, amely bekéri azt, hogy hány évre, és mekkora összeget szeretnénk egy képzeletbeli bankban lekötni. Ezután a program olvassa be azt is, hogy mennyi a lekötés kamata, majd számítsa ki, hogy a megadott év elteltével mennyi pénzt kaphatunk a betéteink után!

4.21. feladat (Futó sebessége – *szint: 2*). Készítsünk programot, amely az alábbi számításokat valósítja meg. Egy százméteres futás résztvevője a táv feléig egyenletesen gyorsul, majd az utolsó tíz méteren egyenletesen lassul.

A program kérje be a futó kezdő sebességét (m/s) egy adott intervallumon belül (3.00 - 5.00), és írja ki tíz méterenként a futó aktuális sebességét km/h-ban!

4.22. feladat (Számok sorozata – *szint: 2*). Írjunk konzolos alkalmazást a 4.22 példaprogram alapján, amely bekér egy egész számot, majd mindaddig kér be további egész számokat, amíg nullát nem kap. A program határozza meg és írja ki a megadott egész számok közül a legnagyobbat.

4.23. feladat (Öttel osztható – *szint: 1*). Írjon programot a 4.23 példaprogram felhasználásával, mely beolvassa a billentyűzetről egy intervallum kezdő és végértéket, majd kiírja a képernyőre az intervallumba eső egész számok közül azokat, melyek 5-tel oszthatók!

A fejezet forráskódjai

4.1. forráskód. Tömb bekérése

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         int osszeg = 0;
         int paros = 0;
         int paratlan = 0;
         for (int i = 0; i < 99; i++)
         {
            Console.Write
             ("Kérem adjon meg egy számot: ");
            int szam =
             int.Parse(Console.ReadLine());
            osszeg = osszeg + szam;
            if (szam % 2 == 0) paros++;
            if (szam % 2 != 0) paratlan++;
            if (osszeg >= 100) break;
         }
         Console.WriteLine ("{0} páros és {1} páratlan..."
                        ,paros,paratlan);
         Console.ReadLine();
      }
   }
}
```

4.2. forráskód. Mátrix bekérése

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace a
{
   class Program
   {
      static void Main(string[] args)
      {
```

```
         int[,] matr = new int[2, 2];
         for (int i=0;i<2;i++)
            for (int j = 0; j < 2; j++)
            {
               Console.Write
               ("Kérem az {0}. sor {1}. elemét: ", i + 1, j + 1);
               matr[i, j] = int.Parse(Console.ReadLine());
            }
         Console.WriteLine();
         Console.WriteLine("A mátrix:");
         Console.WriteLine();
         for (int i = 0; i < 2; i++)
         {
            for (int j = 0; j < 2; j++)
               Console.Write("{0}    ", matr[i, j]);
            Console.WriteLine();
         }
         Console.WriteLine();
         int det = matr[0, 0] * matr[1, 1] - matr[0, 1] * matr[1, 0];
         Console.WriteLine("A mátrix determinánsa: {0}", det);
         Console.ReadLine();
      }
   }
}
```

4.3. forráskód. Négyszögek kirajzolása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace _b
{
   class Program
   {
      static void Main(string[] args)
      {
         int i = 0, x = 0, y = 13;
         Random rnd = new Random();
         int ran;
         while (x < 79){
            i = 0;
            ran = rnd.Next(2, 10);
            while (i < ran && x < 79){
               i++;x++;
               Console.SetCursorPosition(x, y);
               Console.Write("∘");
            }
            i = 0;
            while (i < 8 && x < 79){
               i++;y--;
               Console.SetCursorPosition(x, y);
               Console.Write("∘");
            }
            ran = rnd.Next(2, 10);
            i = 0;
            while (i < ran && x < 79){
               i++;x++;
               Console.SetCursorPosition(x, y);
               Console.Write("∘");
            }
            i = 0;
            while (i < 8 && x < 79){
               i++;y++;
               Console.SetCursorPosition(x, y);
               Console.Write("∘");
            }
         }
         Console.ReadLine();
      }
   }
}
```

4.4. forráskód. A törtes feladat

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace _2c
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("A tört számlálója: ");
         int a = int.Parse(Console.ReadLine());
         int x = a;
         Console.Write("A tört nevezője: ");
         int b = int.Parse(Console.ReadLine());
         int y = b;
         while (a != 0 && b != 0)
         {
            if (a > b)
               a = a-b;
            else
               b = b-a;
         }
```

```
            int lnko = Math.Max(a, b);
            Console.WriteLine();
            Console.WriteLine
             ("Az egyszerűsített tört számlálója: {0}", x / lnko);
            Console.WriteLine
             ("Az egyszerűsített tört nevezője: {0}", y / lnko);
            Console.ReadLine();
         }
      }
   }
```

4.5. forráskód. Karakterek bekérése

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace _2d
{
   class Program
   {
      static void Main(string[] args)
      {
         int i=1;
         Console.WriteLine("Négyzetszámok 1 és 10000 között:");
         Console.WriteLine();
         while (i * i < 10001)
         {
            Console.Write("{0}, ", i * i);
            i++;
         }
         Console.WriteLine();
         Console.WriteLine();
         Console.WriteLine("Köbszámok 1 és 10000 között:");
         Console.WriteLine();
         i = 1;
         while (i * i * i < 10001)
         {
            Console.Write("{0}, ", i * i * i);
            i++;
         }
         Console.ReadLine();
      }
   }
}
```

4.6. forráskód. Összeg kiszámítása

```
static void Main(string[] args)
{
   int s = 0;
   Console.Write("Add meg a k pozitiv számot:");
   int k = Convert.ToInt32(Console.ReadLine());
   for (int i = 1; i <= k; i++)
   {
      s = s + i * (i + 1);
   }
   Console.WriteLine("Osszeg: {0}",s);
   Console.ReadLine();
}
```

4.7. forráskód. Háromszög kirajzolása

```
static void Main(string[] args)
{
   Console.Write ("Add meg az a:");
        int a = int.Parse (Console.ReadLine ());
   string s = "";

   for (int i = 1; i <= a; i++)
   {
      Console.SetCursorPosition(2*a-1 , i);
      s = s + "*";
      Console.Write(s);

   }
   Console.ReadLine();
}
```

4.8. forráskód. Betűk kirajzolása

```
static void Main(string[] args)
{
   Console.Write("Add meg hányszor ismételjem:");
   int n = int.Parse(Console.ReadLine());
   string s = "";

   for (int i = 1; i <= n; i++)
      s = s + "XO";
   Console.Write(s);
   Console.ReadLine();
}
```

4.9. forráskód. Betűkígyó

```
using System;

namespace betukigyo
{
   class Program
   {
      static void Main(string[] args)
      {
         Random rnd = new Random();
         Console.Title = "BETÜKÍGYÓ";
         char[] kigyo = new char[80];
         int i, j, kdb;
         kdb = 0;
         ConsoleKeyInfo gomb = new ConsoleKeyInfo();
         Console.BackgroundColor = ConsoleColor.Red;
         Console.ForegroundColor = ConsoleColor.Green;
         Console.WriteLine("50 KAREKTERBŐL ÁLLÓ BETÜKÍGYÓ");
         do{
            kigyo[kdb] = Convert.ToChar(rnd.Next
              (Convert.ToInt32('A'), Convert.ToInt32('K') + 1));
            kdb++;
            Console.SetCursorPosition(0, 5);
            for (i = 0; i < kdb; i++)
                 Console.Write("{0}", kigyo[i]);
            for (i = kdb; i < 50; i++) Console.Write(" ");
            Thread.Sleep(100);
            while (Console.KeyAvailable){
               gomb = Console.ReadKey(true);
               for (i = 0; i < kdb; i++){
                if (Char.ToUpper(gomb.KeyChar) == kigyo[i]){
                      for (j = i; j < kdb - 1; j++)
                         kigyo[j] = kigyo[j + 1];
                      kdb--;i--;
                   }
               }
            }
         }
         while ((kdb < 50) && (kdb > 0) &&
           (gomb.Key != ConsoleKey.Escape)); Console.WriteLine();
         if (gomb.Key != ConsoleKey.Escape){
            if (0 == kdb) Console.Write("NYERTÉL");
            else Console.Write("NYERTEM");
            Console.WriteLine("\n\nEnterre kilépek!");
            Console.ReadLine();
         }
      }
   }
}
```

4.10. forráskód. Átváltás tízes számrendszerbe

```
using System.Text;

namespace feladat
{
   class feladat
   {
      static void Main(string[] args)
      {
         Console.Write("Bitsorozat : ");
         string bitsor= Console.ReadLine();
         double osszeg=0;
         double hatvany = 2;
         for (int i = 0; i < bitsor.Length; i++)
         {
            hatvany=Math.Pow(2,bitsor.Length-i-1);
            string szamjegy = bitsor.Substring(i, 1);
            osszeg +=int.Parse(szamjegy)*hatvany;
         }
         Console.WriteLine("Az átváltás eredménye : {0}", osszeg);
         Console.ReadLine();
      }
   }
}
```

4.11. forráskód. C# logó kirajzolása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace feladat
{
   class feladat
   {
      static void Main(string[] args)
      {
         int meret=10;
         int vkozep = Convert.ToInt16(40 - meret / 2);
         int fkozep = Convert.ToInt16(12 - meret / 2);
         int harmad = Convert.ToInt16(meret / 3);
         for (int i=0;i<meret;i++){
            Console.SetCursorPosition(vkozep+i, fkozep);
            Console.Write("X");
            Console.SetCursorPosition(vkozep-1, fkozep+i+1);
            Console.Write("X");
            Console.SetCursorPosition(vkozep+i, fkozep+meret + 1);
            Console.Write("X");
            Console.SetCursorPosition
            (80-vkozep + i+harmad*2+1, fkozep + harmad + 1);
```

```
            Console.Write("X");
            Console.SetCursorPosition
            (80 - vkozep + i + 2*(harmad) , fkozep + harmad*2 + 2);
            Console.Write("X");
         }
         for (int i = 0; i < harmad; i++){
            Console.SetCursorPosition(80-vkozep+meret, fkozep+i+1);
            Console.Write("X");
            Console.SetCursorPosition
            (80 - vkozep + meret-1, fkozep +harmad+ i+2);
            Console.Write("X");
            Console.SetCursorPosition
            (80 - vkozep + meret - 2, fkozep + harmad*2 + i + 3);
            Console.Write("X");
            Console.SetCursorPosition
            (80 - vkozep + meret+harmad+1, fkozep + i + 1);
            Console.Write("X");
            Console.SetCursorPosition
            (80 - vkozep + meret + harmad, fkozep + harmad + i + 2);
            Console.Write("X");
            Console.SetCursorPosition
            (80 - vkozep + meret + harmad -1,
                                 fkozep + harmad * 2 + i + 3);
            Console.Write("X");
         }
         Console.ReadLine();
      }
   }
}
```

4.12. forráskód. Csoportosítási feladat

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace feladat_2
{

   class feladat_2
   {
      static void Main(string[] args)
      {
      Console.WriteLine("Véletlenszerű csoport kialakítása...");
      Console.WriteLine("Osztály létszáma!");
      int letszam = int.Parse(Console.ReadLine());
      Console.WriteLine("Max létszám:");
      int csop=int.Parse(Console.ReadLine());
      bool[] osztaly=new bool[letszam];
      Random rnd = new Random();
      int db = 0;
      while (db < letszam){
         int i = rnd.Next(0, letszam);
         if (osztaly[i] == false){
            double c=db/csop;
            Console.WriteLine("A(z) {0}. csoport tagjai:" +
                        "{1}. tanuló", Math.Floor(c)+1,i + 1);
            osztaly[i] = true;
            db++;
         }
      }
      Console.ReadLine();
    }
  }
}
```

4.13. forráskód. Monogrammos feladat

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Feladat_6
{
   class feladat_6
   {
      static void Main(string[] args)
      {
         Console.Write("Kérem, gépeljen be egy nevet: ");
         string nev1 = Console.ReadLine();
         nev1 = ' ' + nev1;
         string nev=nev1.ToUpper();
         string monogram="";
         for (int i = 0; i < nev.Length-2; i++)
         {
            if (nev[i]==' ')
            {
               string betu = nev.Substring(i+1, 2);
               switch (betu)
               {
                  case "CS": monogram += betu + ' ';
                     break;
                  case "DZ": if (nev[i + 3] == 'S')
                           monogram += betu + 'S' + ' ';
                          else monogram += betu + ' ';
                     break;
                  case "GY": monogram += betu + ' ';
                    break;
                  case "LY": monogram += betu + ' ';
```

```
                    break;
                case "NY": monogram += betu + ' ';
                    break;
                case "SZ": monogram += betu + ' ';
                    break;
                case "TY": monogram += betu + ' ';
                    break;
                case "ZS": monogram += betu + ' ';
                    break;
                default: monogram +=
                      nev.Substring(i + 1, 1) + ' ';
                    break;
            }
          }
        }
        Console.WriteLine(monogram);
        Console.ReadLine();
    }
  }
}
```

4.14. forráskód. Összegzést végző feladat

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace szamjegyekosszege
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("Kerek egy szamot: ");
         string szam = Console.ReadLine();

         int osszeg = 0;

         for (int i = 0; i < szam.Length; i++)
         {
            osszeg = osszeg + int.Parse(szam[i].ToString());
         }

         Console.WriteLine("A szamjegyek osszege: {0}.", osszeg);
         Console.ReadLine();
      }
   }
}
```

4.15. forráskód. Pénzérmés feladat

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace fejvagyiras
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("Hanyszor dobjuk fel a penzt? ");
         int db = int.Parse(Console.ReadLine());

         int fej = 0;
         int iras = 0;

         for (int i = 0; i < db; i++)
         {
            Random veletlen = new Random();
            int dobas = veletlen.Next(0, 100);
            if (dobas % 2 == 0)
            {
               fej++;
            }
            else
            {
               iras++;
            }
         }

         Console.WriteLine("{0} db fej, {1} db iras.", fej, iras);
         Console.ReadLine();
      }
   }
}
```

4.16. forráskód. Kukacos feladat

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace feladat2_3
{
   class Program
   {
      static void Main(string[] args)
      {
         ConsoleKeyInfo beolvasott;
         int x = 1;
         int y = 1;
         while (true){
            Kepernyo(x, y);
            beolvasott = Console.ReadKey();
            switch (beolvasott.Key){
               case ConsoleKey.LeftArrow:
                if (x != 1){x--;} break;
               case ConsoleKey.RightArrow:
                if (x != 80){x++;} break;
               case ConsoleKey.UpArrow:
                if (y != 1){y--;} break;
               case ConsoleKey.DownArrow:
                if (y != 23){y++;}break;
               default:break;
            }
         }
      }

      static void Kepernyo(int x, int y)
      {
         Console.Clear();
         Console.WriteLine("(x: {0}; y: {1})", x, y);
         for (int i = 1; i < y; i++){
            Console.WriteLine();
         }
         for (int i = 1; i < x; i++){
            Console.Write(" ");
         }
         Console.Write("@");
      }
   }
}
```

4.17. forráskód. Napszakok feladat

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace feladat
{
   class Program
   {
      static void Main(string[] args)
      {
         DateTime ido = DateTime.Now;
         Console.WriteLine("Jelenlegi ido: {0}:{1}.",
                     ido.Hour, ido.Minute);
         if (ido.Hour > 5 && ido.Hour < 9){
            Console.WriteLine("Jo reggelt!");
         }
         else if (ido.Hour > 8 && ido.Hour < 19){
            Console.WriteLine("Jo napot!");
         }
         else{
            Console.WriteLine("Jo ejszakat!");
         }
         Console.ReadLine();
      }
   }
}
```

4.18. forráskód. Sorozat maximum eleme

```
namespace cl
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Kérem a számokat!");
         int n=-1, max = 0;
         while (n != 0)
            {
               n = Convert.ToInt32(Console.ReadLine());
               if (n > max)
               max = n;
            }
         Console.WriteLine("A legnagyobb szám: {0}" ,max );
         Console.ReadLine();
      }
   }
}
```

5. fejezet - Számok és sorozatok (szerző: Király Roland)

Tartalom

5.1. feladat (Páros és páratlan számok darabszáma – *szint: 2*). Írjunk az 5.1 programhoz hasonló alkalmazást, amelyben kérjünk be *N* darab természetes számot. Az adatok beolvasása után a program írja ki a páros és páratlan számok darabszámát, és a páratlan számok összegét a megadott *N*-ig!

Magyarázat: Amennyiben nem akarunk a bekéréssel bajlódni, elsőként kérjük be az *N* értékét, és addig ne lépjünk ki a beolvasást végző ciklusból, amíg az *N*-szer le nem futott.

Ennél kicsit barátságosabb megoldás, ha addig folytatjuk a beolvasást, amíg a felhasználó le nem nyomja a kilépés gombját, amit mi választunk meg. Ekkor *N* értékérének a bekérésére nem is lesz szükségünk, mivel a beolvasásokat számolhatjuk egy változóban.

5.2. feladat (Számok szorzata – *szint: 1*). Írjon az 5.2 forráskód alapján programot, mely kiszámolja az első *N* szám szorzatát! Az *N* értékét a felhasználó adja meg.

5.3. feladat (Kilométerkövek – *szint: 2*). Készítsünk az 5.3 példához hasonló programot, amely az országúton haladva látott fákat számolgatja. Természetesen pusztán kedvtelésből…A program használója megadhatja a kiindulási pozícióját, valamint a cél pozíciót, mindkettőt kilométerben.

A program virtuális terében, a képzeletbeli út egyik oldalán 3 m-enként, míg a másik oldalán 5 m-enként vannak fák. Adjuk meg az út során azokat a pozíciókat, ahol az út mindkét oldalán fa található.

Magyarázat: Az egyszerűség kedvéért az út mindig nulláról kezdődik, és a fák is a 0. pozíciótól kezdve helyezkednek el három, valamint öt méterenként.

5.4. feladat (Kiírás ciklusokkal – *szint: 1*). Készítsen konzolos alkalmazást az 5.4 alapján, amely teleírja a konzol képernyőt csillagokkal, sorról-sorra oda - vissza haladva. Használhat késleltetést is, hogy az eredmény szemléletesebb legyen. A kiírás a bal felső sarokból kezdődjön, és onnan haladjon jobbra, és lefelé, ahogy a folyó írás is halad…

5.5. feladat (Kiírás ciklusban - továbbfejlesztett változat – *szint: 1*). Írj programot, mely teleírja a konzolképernyőt csillagokkal sorról sorra karakterről karakterre oda - vissza, majd az utolsónak beírt csillagtól kezdve törölje a képernyőt karakterről karakterre haladva fel - le. A kiírás a bal felső sarokból kezdődjön, és jobbra-lefelé tartson. Az 5.5 forrásszövegben láthatunk egy példaprogramot, amely segít a megoldásban.

5.6. feladat (Függvénytábla – *szint: 1*). Készítsünk mini függvénytáblát az 5.6 példaprogram alapján, melyben szerepelnek függőlegesen a pozitív egész számok 1-től - 20-ig, vízszintesen a szám, annak négyzete, és köbe. A táblának legyen fejléce is!

5.7. feladat (Sorozat szorzás nélkül – *szint: 1*). Készítsünk programot az 5.7 példaprogram alapján, amely meghatározza az 1 és 1000 közötti pozitív egész számok szorzatát úgy, hogy nem használhatjuk a szorzás műveletét!

5.8. feladat (Autóverseny – *szint: 2*). Készítsünk programot, mely képzeletbeli autókat versenyeztet. A játékban szereplő két autó 1 és 3 közötti, véletlen számú mezőt tesz meg egy lépésben. Összesen 60 mező áll rendelkezésre a célig. A program írja ki, hogy melyik autó nyert, és a vesztes autó hol tartózkodott a nyertes célba jutásának időpontjában. A versenypályát prezentáljuk csillagokkal ahol a * pontosan 1 mezőt jelent.

5.9. feladat (Kamatos kamatok – *szint: 2*). Írjunk programot, amely a következő problémát oldja meg: 2 gyermek versenyzik, hogy melyik tud többet spórolni. Az egyik havi kamatos kamattal rakja bankba a megspórolt pénzét. A kamatos kamat hozama 3%. A második fix kamatozású, éves lekötésbe fekteti a pénzét. Ennek értéke 7%. A program mondja meg, hogy egy év után kinek lesz több pénze, valamint azt, hogy 10 év után ki, és mennyivel jár jobban.

Magyarázat: Ha általánosabbra szeretnénk megírni a programot, készítsük el úgy, hogy a felhasználó megadhatja a kamat kiszámításához szükséges évek számát is.

5.10. feladat (Vektor feltöltése – *szint: 1*). Írjunk olyan programot, amely véletlen, két számjegyű értékekkel feltölt egy 20 elemű vektort, majd kiírja a képernyőre egymás mellé, vesszővel elválasztva az elemeket.

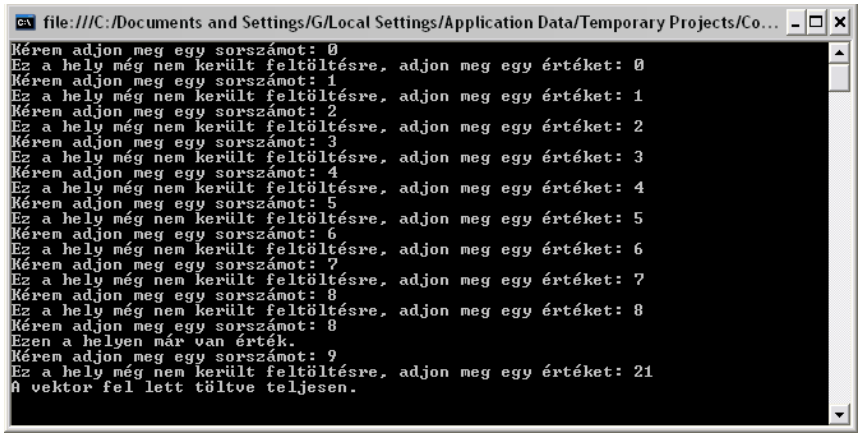
A kiírás után addig kérjünk be két egész számot (a, b), amíg az 'a' kisebb lesz, mint 'b'. Határozzuk meg, hogy hány olyan tömbelem van, amelyik az [a,b] intervallumba esik.

5.11. feladat (Sorozat beolvasása extrákkal – *szint: 1*). Írjunk az 5.11 forrásszöveg alapján olyan programot, amely egy 10 elemű vektort a következőképp tölt fel billentyűzetről:

- bekérünk egy sorszámot
- ellenőrizzük hogy létezik-e ilyen vektorelem egyáltalán, és az még nem került feltöltésre.
- Ha ezen sorszámú vektorelem már kapott értéket, akkor azt még egyszer ne engedjük feltölteni
- ha minden rendben van, akkor bekérhetjük az értéket is
- ha a sorszám -1, akkor kérjük be az értéket, és minden olyan tömbelem, amely még nem kapott értéket - annak legyen ez az értéke

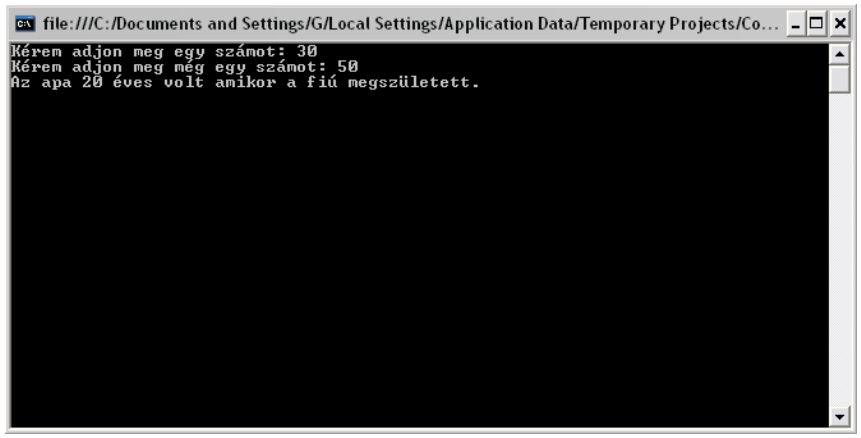
Ezt ismételgessük addig, amíg minden tömbelem meg nem kapta az értékét. Ekkor lépjünk ki a ciklusból, és írjuk ki a vektor elemeit a képernyőre. A példaprogram kimeneti képernyőjét a 5.1 ábrán láthatjuk.

5.1. ábra. Sorozat beolvasása extrákkal



5.12. feladat (Életkorok extra változat – *szint: 1*). Készítsünk az 5.12 példához hasonló alkalmazást, amelyben kérjünk be két egész számot billentyűzetről a 10..90 intervallumból. Amennyiben a beírt számok ezen kívül eső egész számok lennének, úgy addig ismételjük a bekéréseket, amíg megfelelő értékeket nem kapunk. A két számot fogjuk fel mint életkorok, egy apa és a fia életkorait. Adjuk meg, hány éves volt az apa, amikor a fia megszületett.(nem tudni melyik életkort adják meg előbb, a fiút vagy az apáét). Amennyiben az apa fiatalabb volt ekkor mint 18, vagy idősebb mint 50, akkor a program írja ki, hogy „bár ez nehezen hihető”. A program kimeneti képernyője az 5.2 ábrán látható.

5.2. ábra. Életkorok extra változat



5.13. feladat (Legkisebb elem – *szint: 1*). Készítsünk programot, mely egy tetszőleges sorozatról eldönti, hogy melyik a legkisebb, vagy akár a legnagyobb eleme.

Magyarázat: A feladat megoldásához használhatunk egy véletlen számokkal feltöltött tömböt, amit egy for ciklus segítségével bejárunk. Az 5.13 forrásszövegben láthatjuk, hogyan oldhatjuk meg a feladatot, a kimeneti képernyőt az 5.3 ábrán találjuk meg.

Első lépésként azt feltételezzük, hogy a sorozat első elem a legkisebb. Ebben az esetben a második elemtől haladva a sorozat vége felé minden elemre megvizsgáljuk, hogy az kisebb-e a legkisebbnek tekintett elemnél.

Ha igen, akkor ez az elem veszi át a korábbi legkisebb helyét, amennyiben nem, akkor minden marad a régiben.

Igazság szerint ez egy alapvető programozási tétel, melyet minden programozónak ismernie kell. A leíró nyelvi változat a következőképpen néz ki:

```
  minimum = sorozat első elem
ismétlés a sorozat végéig
   ha minimum > sorozat aktuális elem
      minimum = sorozat aktuális elem
   ha vége
   sorozat aktuális elem = sorozat következő elem
ismétlés vége
```

5.3. ábra. Legkisebb elem

5.14. feladat (Legkisebb elemek – *szint: 1*). Készítsünk programot, mely egy tetszőleges sorozatról eldönti, hogy melyik a legkisebb, és a második legkisebb eleme.

Magyarázat: A feladat megoldásához használhatunk egy véletlen számokkal feltöltött tömböt, amit egy for ciklus segítségével bejárunk. Tehát ennek a tömbnek az elemeit kell bejárnunk úgy, hogy minden esetben megvizsgáljuk, melyik a legkisebb elem, ha ennél találunk kisebbet, a korábbi legkisebb elem lesz a második legkisebb, a frissen megtalált elem pedig a legkisebb. Figyelnünk kell továbbá az egyenlőségre.

A minimum kiválasztás programozási tétele a következő:

```
  minimum = sorozat első elem
ismétlés a sorozat végéig
   ha minimum > sorozat aktuális elem
      minimum = sorozat aktuális elem
   ha vége
   sorozat aktuális elem = sorozat következő elem
ismétlés vége
```

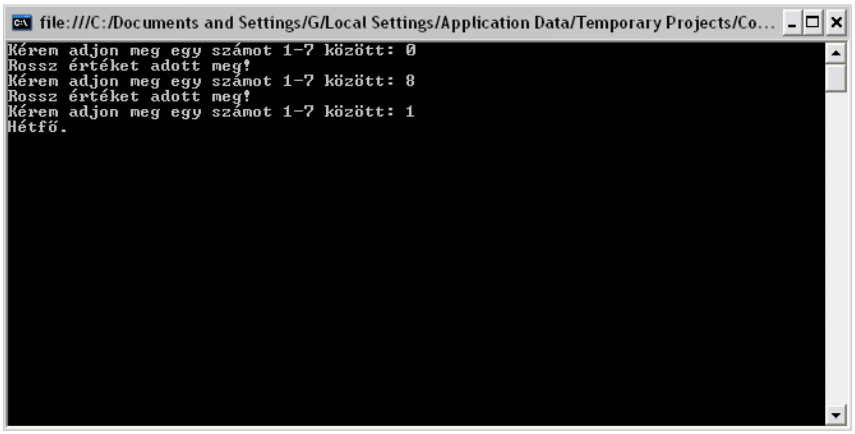
Ezt a programozási tételt kis módosításokkal alkalmazva megkapjuk a feladat megoldását. A módosítás lényege az, hogy be kell vezetnünk egy újabb változót és egy másik elágazást is kell készítenünk.

5.15. feladat (A hét napjai - kiterjesztett változat – *szint: 2*). Készítsünk az 5.15 példához hasonló konzolos alkalmazást, amely paraméterként kap egy egész számot (*int*), majd kiírja a hét azonos sorszámú napját a képernyőre. Az 1-es érték jelenti a hétfőt, a 2-es a keddet, a 7-es a vasárnapot.

Amennyiben a megadott szám nem esik az 1-7 intervallumba, a program írjon hibaüzenetet a képernyőre, majd kérje be újra az számot mindaddig, amíg helyes értéket nem kap. A program kimeneti képét a 5.4 ábrán találjuk meg.

Magyarázat: A megoldás feltételes ciklus utasítás használatát igényli. Ha elkészítettük a számlálós ciklussal operáló verziót, gondolkozzunk el azon is, nem lenne-e érdemesebb hátul tesztelő ciklussal elkészíteni a feladatot.

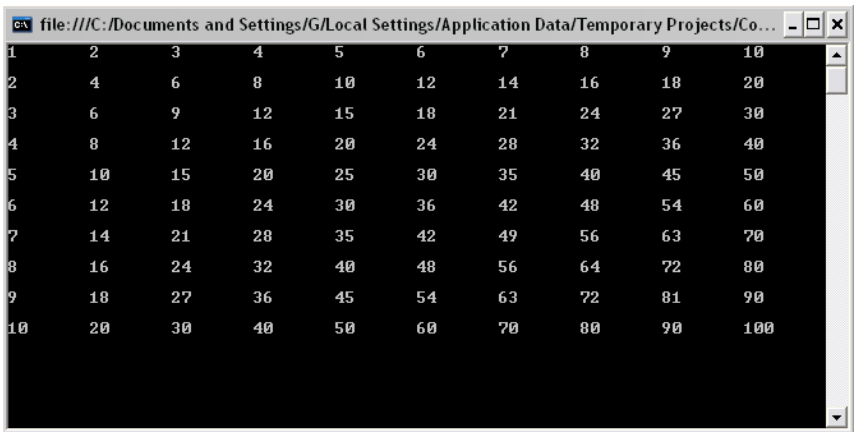
5.4. ábra. A hét napjai - kiterjesztett változat



5.16. feladat (Szorzótábla – *szint: 1*). Készítsünk az 5.5 forrásszöveg alapján konzol programot, mely kiírja a szorzótáblát a képernyőre.

Magyarázat: A szorzótáblát két egymásba ágyazott ciklus segítségével jeleníthetjük meg a képernyőn úgy, hogy a belső ciklus mindig az aktuális sort írja ki, a külső pedig megtöri az adott sort. A belső ciklusban megjelenített számok minden esetben a két ciklus változójának a szorzatából áll elő. A kimeneti képernyő az 5.5 ábrán látható.

5.5. ábra. Szorzótábla



5.17. feladat (Faktoriális kiszámítása – *szint: 1*). Készítsük el az ismert rekurzív faktoriális kiszámítására alkalmas program iteratív változatát, amelynek egy lehetséges változatát az 5.17 példaprogramban láthatjuk!

Magyarázat: A faktoriális kiszámításához használhatjuk a

$$n! = \prod_{i=1}^{10} k$$

képletet.

A formula minden `n>0` egész szám esetén alkalmazható, és ez alapján a

$$4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$$

(a következő érték a 120, majd ezt követi a 720).

Információ: A matematikában egy n nemnegatív egész szám faktoriálisának az n-nél kisebb vagy egyenlő pozitív egész számok szorzatát nevezzük.

A faktoriális értéke nagyon gyorsan növekszik, a 70! értéke: 11 978 571 669 969 891 796 072 783 721 689 098 736 458 938 142 546 425 857 555 362 864 628 009 582 789 845 319 680 000 000 000 000 000 (forrás: Wikipedia)

A faktoriális kiszámítását használják a kombinatorikában, de ugyanúgy a biológiában és a matematika számos területén, mint a deriválás, a Taylor sorok, valamint a binomiális együtthatók kifejezésénél ($\frac{n!}{k!(n-k)!} = \binom{n}{k}$).

A fejezet forráskódjai

5.1. forráskód. Oszthatósági feladat

```
namespace ciklus2
{
   class Program
   {
      static void Main(string[] args)
      {
         int a, b, i, x;
         Console.WriteLine("Kérek egy alsó határt:");
         a = int.Parse(Console.ReadLine());
         Console.WriteLine("Kérek egy felső határt:");
         b = int.Parse(Console.ReadLine());
         Console.WriteLine("5-tel osztható számok:");
         for (i = a; i <= b; i = i + 1)
         {
            if (i % 5 == 0)
               Console.WriteLine("{0}", i);
         }
         Console.ReadLine();

      }
   }
}
```

5.2. forráskód. Szorzatos feladat

```
namespace cikusok
{
   class Program
   {
      static void Main(string[] args)
      {
         double i, n, sz = 1;
         Console.WriteLine("Kérek egy n számot:");
         n = double.Parse(Console.ReadLine());
         for (i = 1; i <= n; i = i + 1)
         {
            sz = sz * i;
         }
         Console.WriteLine("Az első {0} szám szorzata:{1}", n, sz);
         Console.ReadLine();
      }
   }
}
```

5.3. forráskód. Kilométerkövek

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace con
{
 class fák{
      static void Main(string[] args)
      {
         int a, b, i, km;
         Console.Write("Innen indulok: ");
         a = int.Parse(Console.ReadLine());
         Console.WriteLine();
         Console.Write("Ide érkezem: ");
         b = int.Parse(Console.ReadLine());
         Console.WriteLine();
         Console.Write("Itt lesznek mindkét oldalon a fák: ");
         if (a < b){
            km = a;
            for (i = 0; i <= (b - a); i++){
               if ((km % 5 == 0) && ((km % 3 == 0))){
                  Console.Write("{0}, ", km);
                }
               km++;
            }
           }
         else{
            km = b;
            for (i = 0; i <= (a - b); i++){
               if ((km % 5 == 0) && ((km % 3 == 0))){
```

```
                Console.Write("{0} , ", km);
              }
              km++;
            }
          }
          Console.ReadLine();
      }
   }
}
```

5.4. forráskód. Kiírás ciklussalk

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace con
{
   class csillag1
   {
      static void Main(string[] args)
      {
         Console.Clear();
         for (int y = 0; y <= 24; y++)
         {
            for (int x = 0; x <= 79; x++)
            {
               if (y % 2 == 0) Console.SetCursorPosition(x, y);
               else Console.SetCursorPosition(79 - x, y);

               if (y == 24 && x == 79)
                Console.SetCursorPosition(x - 1, y);

               Console.Write("*");
               System.Threading.Thread.Sleep(5);
            }
         }
         System.Threading.Thread.Sleep(3000);
      }
   }
}
```

5.5. forráskód. Kiírás ciklussal - továbbfejlesztve

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace _2._2_feladat
{
   class csillag2
   {
      static void Main(string[] args)
      {
         Console.Clear();
         int x;
         int y;
         for (y = 0; y <= 24; y++){
          for (x = 0; x <= 79; x++){
             if (y % 2 != 0){
               Console.SetCursorPosition(79 - x, y);}
             else{
               Console.SetCursorPosition(x, y);
             }
             if (y == 24 && x == 79)
               Console.SetCursorPosition(x - 1, y);
             Console.Write("*");
             System.Threading.Thread.Sleep(20);
            }
         }
         for (x = 79; x >= 0; x--){
          for (y = 24; y >= 0; y--){
             if (x % 2 != 0){ Console.SetCursorPosition(x, y);
             }
             else {Console.SetCursorPosition(x, 24 - y);}
             if (y == 24 && x == 79)
                  Console.SetCursorPosition(x, y - 1);
             Console.Write(" ");
             System.Threading.Thread.Sleep(20);
            }
         }
         System.Threading.Thread.Sleep(3000);
      }
   }
}
```

5.6. forráskód. Függvénytáblás feladat

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace con
```

```
{
   class fugvenyt
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Szám  Négyzete  Köbe");
         Console.WriteLine("---------------------------");
         for (int i = 1; i <= 20; i++)
         {
            Console.SetCursorPosition(1, i * 2);
            Console.Write("{0}", i);
            Console.SetCursorPosition(13, i * 2);
            Console.Write("{0}", i * i);
            Console.SetCursorPosition(24, i * 2);
            Console.WriteLine("{0}", i * i * i);
            Console.WriteLine("---------------------------");
         }
         Console.ReadLine();
      }
   }
}
```

5.7. forráskód. Szorzat kiszámítása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace feladat
{
   class szorzat
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Add meg a szorzat tényezőit 1 -" +
                     " 1000 közötti egész számokat");
         Console.Write("\n1. tényező: ");
         int a = int.Parse(Console.ReadLine());
         Console.Write("2. tényező: ");
         int b = int.Parse(Console.ReadLine());
         int szorzat = 0;
         for (int i = 1; i <= b; i++)
         {
            szorzat = szorzat + a;
         }
         Console.Write("\nSzorzatuk: {0}", szorzat);
         Console.ReadLine();
      }
   }
}
```

5.8. forráskód. Sorozat beolvasása extrákkal

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         int[] tomb = new int[10];
         for (int i = 0; i < 10; i++){
            tomb[i] = -1;
         }
         int db=0; int sorszam;
         while (db != 10)
         {
          Console.Write("Kérem adjon meg egy sorszámot: ");
          sorszam = int.Parse(Console.ReadLine());
          if (tomb[sorszam] == -1){
            Console.Write("Ez a hely még nem került feltöltésre," +
                        " adjon meg egy értéket: ");
            tomb[sorszam] = int.Parse(Console.ReadLine());
            db++;
          }
          else Console.WriteLine("Ezen a helyen már van érték.");
          if (db == 10)
            Console.WriteLine("A vektor fel lett töltve teljesen.");
         }
         Console.ReadLine();
      }
   }
}
```

5.9. forráskód. Életkorok extra változat

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
```

```
      static void Main(string[] args)
      {
         int a=0; int b=0;
         bool a_ertek_helyes_e = false;
         bool b_ertek_helyes_e = false;
         while (a_ertek_helyes_e!=true){
            Console.Write("Kérem adjon meg egy számot: ");
            a = int.Parse(Console.ReadLine());
            if (a >= 10 && a <= 90) a_ertek_helyes_e = true;
         }
         while (b_ertek_helyes_e!=true){
            Console.Write
            ("Kérem adjon meg még egy számot: ");
            b = int.Parse(Console.ReadLine());
            if (b >= 10 && b <= 90) b_ertek_helyes_e = true;
         }
         if (a > b){
          int x = a - b;
          if (x < 18 || x > 50)
          {
           Console.WriteLine("Az apa {0} éves volt amikor a fiú" +
                     "megszületett, bár ez nehezen hihető.", x);
          }
          else
           Console.WriteLine("Az apa {0} éves volt amikor a fiú" +
                                      " megszületett.", x);
         }
         else{
          int x = b-a;
          if (x < 18 || x > 50){
           Console.WriteLine("Az apa {0} éves volt amikor a fiú" +
                    " megszületett, bár ez nehezen hihető.", x);
          }
          else
           Console.WriteLine("Az apa {0} éves volt amikor a fiú" +
                                     " megszületett.", x);
         }
         Console.ReadLine();
      }
   }
}
```

5.10. forráskód. Minimum kiválasztása

```
static void Main(string[] args)
{
   int[] sorozat = new int[15];
   Random rnd = new Random();
   for (int i = 0; i < 15; i++)
   {
      sorozat[i] = rnd.Next(1, 101);
   }
   int minimum = sorozat[0];
   for (int g = 1; g < 15; g++)
   {
      if (minimum > sorozat[g])
          minimum = sorozat[g];
   }
   Console.WriteLine("A sorozat lekisebb eleme: {0}", minimum);
   Console.ReadLine();
}
```

5.11. forráskód. A hét napjai - kiterjesztett változat

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("Kérem adjon meg egy számot 1-7 között: ");
         int napkod = int.Parse(Console.ReadLine());
         bool helyes_e_az_ertek = false;
         while (helyes_e_az_ertek != true){
            if (napkod >= 1 && napkod <= 7)
             helyes_e_az_ertek = true;
            else{
               Console.WriteLine("Rossz értéket adott meg!");
               Console.Write("Kérem adjon meg egy számot" +
                         " 1-7 között: ");
               napkod = int.Parse(Console.ReadLine());
            }
         }
         switch (napkod){
            case 1:Console.WriteLine("Hétfő.");break;
            case 2:Console.WriteLine("Kedd.");break;
            case 3:Console.WriteLine("Szerda.");break;
            case 4:Console.WriteLine("Csütörtök.");break;
            case 5:Console.WriteLine("Péntek.");break;
            case 6:Console.WriteLine("Szombat.");break;
            case 7:Console.WriteLine("Vasárnap.");break;
         }
         Console.ReadLine();
      }
   }
}
```

5.12. forráskód. Szorzótábla

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         for (int i = 1; i < 11; i++)
         {
            for (int j = 1; j < 11; j++)
            {
            Console.Write("{0}\t", i * j);
            }
            Console.WriteLine();
         }
         Console.ReadLine();
      }
   }
}
```

5.13. forráskód. Szorzótábla

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         int eredmeny=1;
         Console.Write
           ("Kérem adjon meg egy számot: ");
         int a =
             int.Parse(Console.ReadLine());
         if (a > 0)
         {
            for (int i = 1; i < a+1; i++)
            {
               eredmeny = eredmeny * i;
            }
            Console.WriteLine
               ("A szám faktoriálisa: {0}", eredmeny);
         }
         Console.ReadLine();
      }
   }
}
```

6. fejezet - Vektorokkal és azok kezelésével kapcsolatos feladatok (szerző: Király Roland)

Tartalom

6.1. feladat (Sorozatok – *szint: 1*). Írjunk olyan programot, amely bekéri egy sorozat első négy elemét, majd meghatározza, hogy

- ez a 4 elem számtani sorozatot alkot-e (az elemek különbsége állandó)
- mértani sorozatot alkot-e (az elemek hányadosa állandó)

Magyarázat: A program elkészítéséhez, amelyet a 6.1 forrásszövegben láthatunk használjunk ciklust. Ha még nem használtunk listát, vagy tömböt, a négy elemet tároljuk négy változóban, ellenkező esetben definiáljunk egy 4 elemű tömböt, melybe az elemeket eltárolhatjuk.

A sorozat bejárása során minden elemre megállapítható, hogy milyen az utána következő elemhez a viszonya. Állandó-e a különbségük `t[i]-t[i+1]`, vagy `a - b`. Ugyanez igaz a hányadosra is.

Információ: A számtani sorozat (vagy aritmetikai sorozat egy elemi matematikai fogalom, mely a matematika számos részterületén előfordul.

Egy legalább három számból álló – akár véges, akár végtelen – sorozatot akkor nevezünk számtani sorozatnak, ha a szomszédos elemek különbsége - differenciája - (a sorozatra jellemző) állandó. forrás: Wikipedia.

6.2. feladat (Bináris számjegyek kezelése – *szint: 1*). Készítsünk konzolos alkalmazást a 6.2 példaprogram mintájára, amely beolvas egy bináris számot, majd kiírja, hogy hány 1-es számjegy szerepel a számsorozatban!

Magyarázat: Ha nem beolvasni, hanem generálni szeretnénk a számjegyeket, akkor egy tömb, vagy egy lista lesz a legalkalmasabb a tárolásra. Az egyszerűbb megoldás viszont az, ha bekérjük a számjegyeket a billentyűzetről. Az egyesek megszámlálása ezek után gyerekjáték. Csak definiálnunk kell egy változót, amely segítségével egy ciklusban egyesével számlálhatjuk az egyeseket.

Véletlen számot az alábbi formulával generálhatunk:

```
  Random R = new Random();
int adat = R.next(...);
```

Amennyiben a bonyolultabb megoldást választjuk, figyeljünk a következőkre: Mivel a vektor megfelelője a C# nyelvben a tömb (*típus[db]*). A feladat megoldásához is használhatunk egy *int* típusú elemeket tartalmazó tömböt, ami a következőképpen hozható létre:

```
  int n = 10; //Tetszőleges egész szám,

int[] t = new int[n];
```

A tömb feltöltését egy ciklussal oldhatjuk meg úgy, hogy a ciklusmag minden lefutásakor létrehozunk egy véletlen számot, melyet hozzárendelünk a tömb aktuális indexű eleméhez.

Információ:

A tömb elemeinek létrehozása egy ciklusban történik.

Ha az elemeket ugyanabban a ciklusban dolgoznánk fel, amelyben létrehoztuk őket, akkor sajnálatos módon egy alapvető programtervezési hibát követnénk el. Ne tegyük!

6.3. feladat (Egyedi véletlen számok – *szint: 2*). Fejlesszük tovább a „Véletlen számok” feladatban bemutatott programot úgy, hogy a tömb elemszámát a felhasználó adja meg, valamint ügyeljünk arra is, hogy a tömbbe csak olyan számok kerülhessenek, melyeket előzőleg még nem generáltuk le a véletlen szám generátor segítségével.

Magyarázat: A megoldás több ciklus használatát igényli, melyeket egymásba kell ágyazunk. a megoldást a 6.3 példaprogramban láthatjuk.

Az ellenőrzés lépései:

- Állítsuk elő az aktuális véletlen számot.
- Egy ciklus segítségével járjuk végig az eddig generált számokat, és vizsgáljuk meg, hogy az éppen előállított érték szerepel-e köztük.
- Ha nem, akkor hozzákezdhetünk a következő szám előállításához
- Ha igen, akkor ”dobjuk el” a generált számot, és készítsünk újat

6.4. feladat (Lottó számok – *szint: 1*). Készítsünk konzol programot, mely előállítja egy lottó sorsolás számait!

Magyarázat: A számokat első megközelítésben nem kellene tárolnunk, de természetesen a Lottó számok nem ismétlődhetnek.

Magyarázat: Ezt a feltételt a programunk csak úgy tudja teljesíteni, ha összehasonlítja az éppen generált számot a korábban előállított számok mindegyikével. Amennyiben talál egyezést, úgy a számot újra generálja. Ehhez a megoldáshoz mindenképpen valamilyen vektor típusú adatszerkezetet kell használnunk.

A megoldáshoz, amit a 6.4 forrásszövegben láthatunk, mindezek mellett véletlen szám generátort kell használni, ami 1-90 közé eső számokat állít elő.

6.5. feladat (Konzolos menüpontok kezelése – *szint: 2*). Készítsünk egy egyszerű parancssori programot, mely néhány menüponttal rendelkezik. A menüpontok kiírására használjuk a *Console* kiíró utasításait.

A menü a következőképp nézzen ki:

```
1 Első menüpont
2 második menüpont
3 Harmadik menüpont
4 Negyedik menüpont
5 Kilépés
```

Magyarázat: A program közvetlenül az elindítása után írja ki a menüket a képernyőre, ahogy ezt a 6.5 forrásszövegben láthatjuk, majd olvasson be egy karaktert. Amennyiben a beolvasott szám az 1-4 intervallumba esik, úgy a képernyőre íródjon ki, hogy melyik menüpont került kiválasztásra.

Ezt a műveletet addig kell ismételni, míg a felhasználó az 5-ös számot nem adja meg. Ez esetben a program fejezze be a futását.

Az elkészítés során alkalmazzuk a *switch* vezérlő szerkezetet, melyet egy hátul tesztelő ciklusban helyezünk el. A ciklusnak mindaddig kell futnia, amíg a beolvasott érték nem egyenlő 5-tel.

6.6. feladat (Átváltás kettes számrendszerbe – *szint: 2*). Készítsünk programot, mely tetszőleges tízes számrendszer beli egész számot átvált kettes számrendszerbe. Az átváltandó számot a billentyűzetről olvassuk be, majd az átváltás eredményét ugyanide írjuk ki (lásd: 6.6).

Magyarázat: Az átváltás a legkönnyebben úgy végezhetjük el, ha az átváltandó számot osztjuk a számrendszer alapszámával, vagyis a kettővel. Az osztás maradéka a kettes számrendszerbeli szám lesz, az egészrészt pedig tovább kell osztanunk mindaddig, amíg el nem érjük a nullát.

A megoldáshoz használjunk ciklust. A ciklus magjában el kell végeznünk a maradékos, majd az egész osztást és el kell tárolnunk a maradékokat egy listában, vagy egyéb tetszőleges adattípusban.

A ciklus lefutása, és a számítások elvégzése során megkapjuk a kettes számrendszer beli számot.

6.7. feladat (Átváltás tetszőleges számrendszerekbe – *szint: 3*). Készítsünk olyan programot, amely tízes számrendszerbeli egész számokat átvált tetszőleges, a felhasználó által megadott számrendszerbe. Az átváltandó számot, valamint a számrendszer alapszámát a billentyűzetről olvassuk be, majd az átváltás eredményét ugyanide írjuk ki.

Figyeljünk arra is, hogy 9 fölött a számokat az *ABC* nagy betűivel helyettesítjük.

Magyarázat: A megoldáshoz az „Átváltás kettes számrendszerbe” című feladathoz hasonlóan használjunk ciklust. A ciklus magjában el kell végeznünk a maradékos osztást, majd az egész osztást a számrendszer alapszámával, ezután el kell tárolnunk a maradékokat egy listában, vagy egyéb tetszőleges adattípusban.

A ciklus lefutása után megkapjuk a kettes számrendszerbeli számjegyeket. Figyelem! Ügyeljünk arra is, hogy a számrendszer alapszámánál ne lehessen megadni nullát, vagy negatív számot. A nullával való osztás egyébként is komoly hibát eredményez.

6.8. feladat (Azonos elemek – *szint: 1*). Készítsünk a 6.8 példaprogram alapján konzolos alkalmazást, amelyben létrehozunk egy n-elemű tömböt, melybe véletlen számokat helyezünk el. Állapítsuk meg, hogy a generált számok egyformák-e, vagy különbözőek.

Magyarázat: A program megoldásához használjuk valamely ismert ciklust, ami bejárja az előzőleg generált listát. Ahhoz, hogy eldöntsük, minden szám egyforma-e, elég a bejárás elején egy változó értékét egyre állítani, és addig nem változtatni, az értékén, amíg ugyanolyan számokat találunk, mint az első elem.

Ha a ciklus végére sem módosult a változó értéke, akkor a sorozat teljesen homogén, ellenkező esetben nem az.

A teszteléshez készítsünk egy olyan tömböt is, amely egyforma számokat tartalmaz, mert ellenkező esetben a véletlen szám generátorral nagyon sokáig kellene várnunk az ideális esetre.

Információ: A véletlen számok az informatikában nem valódi véletlen számok, mivel bármely mechanikus eljárás, amit a számítógépeken alkalmazunk, egy idő után ismétlődő sorozatokat fog előállítani. Ezt a típusú véletlen számot *pszeudo* véletlen számnak nevezzük.

Amennyiben valódi véletlen számokra van szükségünk, vehetünk „űrzajt" a NASA-tól, vagy a természethez kell fordulnunk segítségért.

> **6.9. feladat (Statisztika listákról – *szint: 1*).** Készítsünk a 6.9 program alapján olyan alkalmazást, amely bekér a felhasználótól egy számot, majd létrehoz egy ilyen hosszúságú tömböt. A tömb elemeit bekéri a billentyűzetről, majd megállapítja, hogy hányféle értéket kapott!

Magyarázat: A kapott elemeket sorban meg kell vizsgálni, és ha valamelyik elemből találunk egyet, a hozzá rendelt számlálót meg kell növelnünk egyel.

Legalább annyi számlálóra van szükségünk, amennyi féle elem előfordulhat, vagyis a tömb elemszámmal azonos számúra, praktikusan egy azonos hosszúságú tömbre.

Információ:

A statisztika a valóság számszerű információinak megfigyelésére, összegzésére, elemzésére és modellezésére irányuló gyakorlati tevékenység és tudomány.

A statisztika alapja sok olyan eljárásnak, amely titkos szövegek, vagy kódok feltörését teszi lehetővé. Kémek százai dolgoznak azon, hogy nagyhatalmak egymásnak, vagy sajátjaiknak küldött üzeneteit karakter statisztikák segítségével feltörjék, és az információt a hasznukra fordítsák.

A karakter statisztikákon alapuló kódfejtés azon a felismerésen alapszik, hogy minden beszélt nyelvben meg lehet határozni a leggyakoribb karaktereket. Amennyiben rendelkezésre áll ez az információ, a karakter statisztika segít abban, hogy megállapítsuk melyik a titkos szövegben leggyakrabban előforduló jel. Ez, és némi fejtörés segít a kódfejtőknek kitalálni a titkot...

Egyes nézetek szerint a statisztikai adatok 5%-os eltéréssel jól mérnek, de ezt a statisztika módszereivel számították ki...

> **6.10. feladat (Karakterek számlálása – *szint: 1*).** Írjunk a 6.10 példaprogram alapján olyan programot, mely a felhasználótól beolvas egy tetszőleges *string* típusú adatot, valamint egy karaktert, majd kiírja a képernyőre, hogy a megadott karakter hányszor szerepel a szövegben!

Magyarázat: A beolvasott szöveget egy ciklus segítségével bejárhatjuk. A ciklus minden lefutásakor meg kell vizsgálnunk, hogy az aktuális karakter megegyezik-e a keresett karakterrel. Amennyiben igen, egy változó értékét meg kell növelnünk egyel.

Információ: Ez a feladat egy programozási tételen alapszik, melynek a neve: megszámlálás tétele. A megszámláláshoz hasonló tételek még az eldöntés, kiválasztás és a kiválogatás tételei, melyek implementációját szintén ciklussal oldjuk meg.

> **6.11. feladat (Szöveges adatok vizsgálata – *szint: 2*).** Készítsünk konzolos alkalmazást, amely eldönti egy tetszőleges szövegről, hogy az *palindrom* típusú-e, vagy sem. A szöveget a billentyűzetről olvassuk be mindaddig, amíg a felhasználó folytatni szeretné a vizsgálódást. A 6.11 példaprogramban megtekinthetjük az egyik lehetséges megoldást.

Magyarázat: A palindrom szövegek előlről és visszafelé olvasva is ugyanazt jelentik.

Római fővezér - rézevő fia, Mór. (Babits Mihály), vagy az ismert: Indul a Gőrög Aludni.

Ahhoz, hogy egy szövegről eldöntsük, hogy megfelel-e ennek a követelménynek, meg kell állapítanunk, hogy az *i*-edik betűje minden $i \in n$ esetén megegyezik-e az $n - i + 1$-edik betűvel. Az *i* az aktuális szövegindex, míg az *n* a szöveg hossza. A vizsgálatot elég a szöveg közepéig elvégezni ($n\%2$).

Információ: Az 1800-as évek végén Breyer Gyula magyar sakk-nagymester írt egy szerelmes levelet, ami oda-vissza olvasva ugyanaz:

„Nádasi K. Ottó Kis-Adán, májusi szerdán e levelem írám. A mottó: Szivedig ime visz irás, kellemest író! Szinlelő sziv rám kacsintál! De messzi visz szemed... Az álmok (ó csaló szirének ezek, ó csodaadók) elé les. Irok ime messze távol. Barnám! Lám e szivindulat Öné. S im e sziv, e vér ezeket ereszti ki: Szivem! Ime leveled előttem, eszemet letevő! Kicsike! Szava remegne ott? - Öleli karom át, Édesem! Lereszket évasziv rám. Szivem imád s áldozni kér réveden - régi gyerekistenem. Les im. Előtte visz szived is. Ég. Érte reszketek, szeret rég és ide visz. Szivet - tőlem is elmenet - siker egy igérne, de vérré kinzod (lásd ám: ime visz már, visz a vétek!) szerelmesedét. Ámor (aki lelőtt ó engem, e ravasz, e kicsi!) Követeltem eszemet tőled!

E levelem ime viszi... Kit szeretek ezer éve, viszem is én őt, aludni viszem. Álmán rablóvá tesz szeme. Mikor is e lélekodaado csók ezeken éri, szól: A csókom láza de messzi visz!... Szemed látni csak már!... Visz ölelni!... Szoríts!... Emellek Sári szivemig. Ide visz Ottó. Ma már ím e levelen ádresz is uj ám. Nádasi K. Ottó Kis-Adán."

Forrás: Wikipedia.

> **6.12. feladat (Szöveg tükrözése – *szint: 1*).** Készítsünk alkalmazást, mely egy tetszőleges szöveget tükröz a középső elemére. Amennyiben a szöveg páros számú karaktert tartalmaz, annak képzeletbeli közepére végezze el a tükrözést.

Magyarázat: A szöveg tükrözése ciklussal oldható meg, ahogy azt a 6.12 programban is láthatjuk.

A ciklust addig kell futtatni, míg el nem érjük a szöveg közepét *egészrész(szöveghossza osztva 2)*. A szöveg végéig végezve a tükrözést sajnos visszakapjuk az eredeti szöveget. A ciklusmagban minden lépésben cseréljük meg az *i*-edik elemet az $n - i + 1$-edik elemmel. Az *n* a szöveg hossza, az *i* a ciklus változója, vagyis az aktuális elem-index a szövegben.

Vigyázat! A C# nyelvben a *string* típus nem engedi meg, hogy az elemeit egyesével felülírjuk.

Amennyiben mindenképpen szeretnénk használni a *string* osztályt, konkatenálni kell az egyes elemeket egy új változóba a $+$ operátor segítségével az alábbi módon:

```
ujszoveg = ujszoveg + szoveg[szoveg.Length-i];
```

Egy másik megoldáshoz használhatunk *StringBuilder*-t, vagy a karaktereket a tükrözés előtt helyezzük el egy tömbben, esetleg egy listában, melynek az elemeit tetszés szerint meg lehet változtatni.

> **6.13. feladat (Szöveg elemzése – *szint: 2*).** Készítsünk konzol programot, mely kiszámítja egy kifejezés értékét. A program a kifejezést szöveges formában kapja meg a billentyűzetről.

A megadott kifejezésre az egyszerűség kedvéért a következő megkötések vonatkoznak:

- Nem tartalmazhat szóközt, vagy egyéb „white-space” karaktereket.
- Nem lehet benne csak egy operátor és két operandus.
- Az operandusok kizárólag egész számok lehetnek.

A 6.13 példaprogramban látottakat felhasználhatjuk a megoldáshoz.

Magyarázat: A beolvasott szöveget bontsuk szét a *string* a típus megfelelő metódusainak használatával. Az `S.IndexOf(A)` az *A S*-ben elfoglalt helyét, vagyis az indexét adja vissza egész számként. Segítségével megkereshetjük az operátor pozícióját a szövegben. Ezután nincs más dolgunk, minthogy az operátort megelőző, és az azt követő *string* darabokat egész számmá konvertáljuk. Így megkapjuk a két operandust is.

Az operátort kiemelve, és megvizsgálva (pl: egy *switch*, vagy egy *if* ágaiban) már el tudjuk végezni a megfelelő műveleteket, és ki tudjuk írni az eredményt a képernyőre.

6.14. feladat (Kifejezést tartalmazó szöveg tisztítása – *szint: 2*). Készítsünk a 6.14 forrásszöveg alapján konzol programot, amely beolvas egy tetszőleges hosszúságú szöveget, melyet azonnal fel is dolgoz. A program a szöveges kifejezést *string* formában kapja meg billentyűzetről. A kifejezésre a következő megkötésnek kellene érvényesülnie:

Nem tartalmazhat szóközt, vagy egyéb „white-space” karaktereket.

Amennyiben találunk szóközöket a szövegben, úgy azokat távolítsuk el. Valójában ez a művelet a program legfontosabb momentuma.

Magyarázat: Ezt a problémát úgy oldhatjuk meg, ha mindaddig cseréljük a dupla szóközöket szimpla szóközre, amíg találunk ilyeneket a szövegben. A keresésre az *IndexOf* metódust, a cserére a *Replace* metódusát használjuk a *string* típusnak.

Információ: A szöveg tísztítása nagyon hasonlít arra a folyamatra, amikor a különböző programozási nyelvek fordító programja beolvassa a programozó által írt sorokat, majd a fölösleges szóközöket, tabulátor jeleket, „kommenteket”, és egyéb nem kívánatos részeket eltávolítja a szövegből.

A fordítóprogram ezen részét „Source handler”-nek, vagy „Input Handler”-nek nevezzük.

6.15. feladat (Szöveg elemeinek cseréje - kiterjesztett megoldás – *szint: 3*). A „Szöveg elemeinek cseréje” programot módosítsuk úgy, hogy a cserélendő szöveget, valamint azt, hogy mire cseréljük ki, a felhasználó adja meg. Ezzel a megoldással bármit ki tudunk cserélni a szövegben bármi másra.

Módosíthatjuk a programunkat úgy is, hogy a cserélendő elemeket, és azokat is, amikre ezeket cseréljük, listákban lehessen elhelyezni.

Magyarázat: A cserélendő elemek listáját egyesével kell bejárnunk, és minden elem esetén el kell végeznünk a cserét, amit az eredeti programban is elvégeztünk. Ekkor a listák bejárásához is ciklusokat kell használnunk, nem csak a cserékhez.

6.16. feladat (Nagy számok összeadása – *szint: 3*). Készítsünk programot, mely tetszőleges nagy egészeket képes összeadni. A „nagy” szó ebben az esetben azt jelenti, hogy a C# nyelvben tárolható legnagyobb egész számoknál is nagyobb számokat legyünk képesek feldolgozni, majd kezelni az eredményt. A 6.16 példaprogramban találjuk a feladat egy lehetséges megoldását.

Magyarázat: Amennyiben szövegesen tároljuk a számokat, bármekkora számot be tudunk kérni a billentyűzetről. Az elvi korlát az, hogy a felhasználónak mikor fárad el a keze a gépelés közben. A számokat úgy tudjuk összeadni, mint ahogy azt papíron is tennénk.

A két számot tartalmazó szövegre listaként tekinthetünk (valójában azok is), és minden elemre külön elvégezhetjük az összeadást úgy, hogy az adott helyen található elemet számmá konvertáljuk.

A másik lista megfelelő elemével megtehetjük ugyanezt, majd a két számot összeadhatjuk. Természetesen az eredményhez minden lépésben hozzá kell adni az előző eredmény 10 fölötti részét is. Az összeadások elvégzése után az eredményt vissza konvertáljuk szöveggé - miután itt is képeztük a megfefelő átvitelt - és elhelyezzük az eredményt a tárolására szánt szövegben.

A listák végére érve megkapjuk az eredményt szintén szöveges formátumban és kiírhatjuk a képernyőre.

Természetesen ez a klasszikus megoldás, amitől eltérhetünk, ha jobb, vagy épp egyszerűbb megoldást szeretnénk létrehozni.

Információ: A nagy számokat rengeteg fontos tudományterületen használjuk. Ilyen terület a prím számok keresése, a különböző titkosítási eljárások és azok alkalmazása, vagy a nagy számokkal dolgozó csillagászati számítások. Az ilyen méretű adatokkal való munka korlátja sajnos nem a tár-terület, hanem az idő, ami alatt a számításokat el lehet végezni. Sok titkosító eljárás titkossága is az idő tényezőn alapszik. Olyan hosszú idő (évek, vagy évtizedek) kell a titkosított adatok feltöréséhez, hogy egyszerűen nem érné meg elkezdeni a munkát.

6.17. feladat (Nagy számok szorzása – *szint: 4*). Készítsünk a 6.17 példaprogram mintájára konzolos alkalmazást, mely tetszőleges nagy számokat képes szorozni egymással. Az összeadni kívánt számokat billentyűzetről olvassuk be, és szöveges formában tároljuk (*string*). A „nagy” szó ebben az esetben azt jelenti, hogy a C# nyelv alaptípusai által ábrázolható legnagyobb egész számoknál feldolgozni, majd kezelni az eredményt.

Magyarázat: Gondoljunk arra, hogyan szorzunk nagy számokat papír és ceruza segítségével! Egymás mellé írjuk a szorzót és a szorzandó számot, majd egyesével, helyi értékek szerint eltolva minden számjegyre nézve elvégezzük el a szorzást. A kapott eredményeket egymás alá írjuk, majd elemenként elvégezzük az összeadásokat ügyelve az átvitel kezelésére.

A programunknak is valami hasonlót kell végeznie. Ha a szorzásra úgy tekintünk, mint összeadások sorozatára, a feladat könnyedén megoldható a „Nagy számok összeadása” című feladatban ismertetett összeadó rutin felhasználásával.

6.18. feladat (Üres területek – *szint: 2*). Állítsunk elő véletlen egész számokat, majd határozzuk meg azt, hogy melyik a leghosszabb nullákat tartalmazó sorozat az előállított listában.

Magyarázat: A feladat megoldásához tömböt, vagy listát alkalmazhatunk (természetesen más adatszerkezetet is), amelybe egy véletlen szám generátor segítségével sok-sok véletlen számot generálunk. Az előállított listát be kell járnunk valamely ismert vezérlő szerkezet segítségével, és minden megtalált nulla után számolnunk kell, hogy azt hány darab nulla követi. A leghosszabb, csak nullákat tartalmazó sorozat helyét, vagyis az első nulla indexét meg kell jegyeznünk mindaddig, amíg hosszabb sorozatot nem találunk.

Ha biztosak vagyunk a tudásunkban, megpróbálhatjuk a feladatot NxM méretű mátrix, vagy NxMxK-s adatszerkezettel is megoldani.

A fejezet forráskódjai

6.1. forráskód. Sorozat vizsgálata

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         int[] tomb = new int[4];
         for (int i = 0; i < 4; i++)
         {
          Console.Write("{0}. elem: ", i+1);
          tomb[i] = int.Parse(Console.ReadLine());
         }
         int elso=tomb[0];
         int allando=tomb[1]-tomb[0];
         int hanyados=tomb[1]/tomb[0];
         bool szamtani_e=true;
         bool mertani_e = true;
         for (int j = 1; j < 4; j++)
         {
            if (elso + (j * allando) != tomb[j]) szamtani_e = false;
            if (tomb[j - 1] * hanyados != tomb[j]) mertani_e = false;
         }
         if (szamtani_e) Console.WriteLine("A sorozat számtani.");
         if (mertani_e) Console.WriteLine("A sorozat mértani.");
         Console.ReadLine();
      }
   }
}
```

6.2. forráskód. Bináris számok

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
      int db=0;
      Console.Write("Kérem adjon meg egy bináris számot: ");
      string szam = Console.ReadLine();
      for (int i = 0; i < szam.Length; i++){
        if (Convert.ToString(szam[i]) == "1") db++;
      }
      Console.WriteLine
      ("A megadott bináris számban {0} darab egyes szerepelt.", db);
      Console.ReadLine();
      }
   }
}
```

6.3. forráskód. Egyedi véletlen számok

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Random rnd = new Random();
         Console.Write("Kérem adja meg a vektor elemszámát: ");
         int elemszam = int.Parse(Console.ReadLine());
         int[] tomb = new int[elemszam];
         int db=0;
         while(db!=elemszam){
            bool egyezike = false;
            int veletlenszam = rnd.Next(1,101);
            for (int i = 0; i < db; i++){
               if (tomb[i] == veletlenszam) egyezike = true;
            }
            if (!egyezike){
               tomb[db] = veletlenszam;
               db++;
            }
         }
         Console.Write("A vektor elemei: ");
         for (int i = 0; i < elemszam; i++){
            Console.Write("{0}, ", tomb[i]);
         }
         Console.ReadLine();
```

```
      }
   }
```

6.4. forráskód. Lottózó programja

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Random rnd = new Random();
         int[] tomb = new int[5];
         int db=0;
         while(db!=5){
            bool egyezike = false;
            int veletlenszam = rnd.Next(1,91);
            for (int i = 0; i < db; i++){
               if (tomb[i] == veletlenszam) egyezike = true;
            }
            if (!egyezike){
               tomb[db] = veletlenszam;
               db++;
            }
         }
         Console.Write("A lottó számok: ");
         for (int i = 0; i < db; i++){
            Console.Write("{0}, ", tomb[i]);
         }
         Console.ReadLine();
      }
   }
}
```

6.5. forráskód. Konzolos menükezelés

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("1. menupont");
         Console.WriteLine("2. menupont");
         Console.WriteLine("3. menupont");
         Console.WriteLine("4. menupont");
         Console.WriteLine("5. menupont kilepes");
         int melyik = 0;
         while (melyik != 5){
            Console.Write("Menupont kodja: ");
            melyik = int.Parse(Console.ReadLine());
         switch (melyik){
          case 1:Console.WriteLine
               ("Az elso menupontot valasztotta ki.");break;
          case 2:Console.WriteLine
               ("A masodik menüpontot valasztotta ki.");break;
          case 3:Console.WriteLine
               ("A harmadik menupontot valasztotta ki.");break;
          case 4:Console.WriteLine
               ("A negyedik menupontot valasztotta ki.");break;
          case 5:Console.WriteLine
              ("A kilepes menupontot valasztotta ki.");break;
         }
         }
         Console.ReadLine();
      }
   }
}
```

6.6. forráskód. Átváltás kettes számrendszerbe

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ConsoleApplication1
{
   class Program{
      static void Main(string[] args)
      {
         Console.Write ("10-es számrendszerbeli szám:");
         int a = int.Parse(Console.ReadLine());
         List<int> binaris_szam = new List<int>();
         while (a != 0){
```

```
            if (a % 2 == 0){
               a = a / 2;
               binaris_szam.Add(0);
            }
            else{
               a = (a - 1) / 2;
               binaris_szam.Add(1);
            }
         }
         Console.Write("A bináris szám: ");
         for (int i = 0; i < binaris_szam.Count; i++ ){
            Console.Write
            (binaris_szam[binaris_szam.Count-i-1]);
         }
         Console.ReadLine();
      }
   }
}
```

6.7. forráskód. Azonos elemek

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         int [] tomb = new int[10];
         int db = 0;
         bool van=false;
         Random rnd = new Random();
         for (int i = 0; i < 10; i++)
         {
            tomb[i] = rnd.Next(1, 101);
            db++;
            if (db > 1)
            {
               for (int j = 0; j < db-1; j++)
               {
                  if (tomb[i] == tomb[j]) van = true;
               }
            }
         }
         if (van) Console.WriteLine
            ("A tömb tartalmaz azonos elemet.");
         Console.ReadLine();
      }
   }
}
```

6.8. forráskód. Statisztika listákról

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("Kérem adja meg a vektor méretét: ");
         int meret = int.Parse(Console.ReadLine());
         int [] tomb = new int[meret];
         int most_ennyi_elem = 0;
         int db=1;
         for (int i = 0; i < meret; i++)
         {
            Console.Write("Kérem adja meg
                      a tömb {0}. elemét: ", i+1);
            tomb[i] = int.Parse(Console.ReadLine());
            most_ennyi_elem++;
            if (most_ennyi_elem > 1)
            {
               for (int j = 0; j < most_ennyi_elem-1; j++)
               {
                  if (tomb[i] == tomb[j])
                  {
                     db = db + 1;
                     break;
                  }
               }
            }
         }
         Console.WriteLine
         ("A tömb {0} nem azonos elemet tartalmaz.",meret-db+1);
         Console.ReadLine();
      }
   }
}
```

6.9. forráskód. Karakterek számlálása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("Kérem adjon meg egy stringet: ");
         string szoveg = Convert.ToString(Console.ReadLine());
         Console.Write("Adja meg a keresni kívánt karaktert: ");
         string keresett_karakter =
                  Convert.ToString(Console.ReadLine());
         int i = 1;
         int db = 0;
         while (i < szoveg.Length)
         {
            if (Convert.ToString(szoveg[i]) ==
                        keresett_karakter) db++;
            i++;
         }
         Console.WriteLine("A stringben {0} darab
                  keresni kívánt karakter található.", db);
         Console.ReadLine();
      }
   }
}
```

6.10. forráskód. Szavak vizsgálata

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         bool palindrom = true;
         Console.Write("Kérem adjon meg egy szöveget: ");
         string szoveg = Convert.ToString(Console.ReadLine());
         int i = 1;
         int fele;
         if (szoveg.Length % 2 != 0) fele = (szoveg.Length - 1) / 2;
         else fele = szoveg.Length / 2;
         while (i < fele)
         {
            if (Convert.ToString(szoveg[i-1]) !=
         Convert.ToString(szoveg[szoveg.Length - i]))
            palindrom = false;
            i++;
         }
         if (palindrom) Console.WriteLine("A szöveg palindrom.");
         else Console.WriteLine("A szöveg nem palindrom.");
         Console.ReadLine();
      }
   }
}
```

6.11. forráskód. Szöveg tükrözése

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("Kérem adjon meg egy szöveget: ");
         string szoveg = Console.ReadLine();
         string ujszoveg = "";
         for (int i = 1; i<szoveg.Length+1 ; i++)
         {
            ujszoveg = ujszoveg + szoveg[szoveg.Length-i];
         }
         Console.WriteLine(ujszoveg);
         Console.ReadLine();
      }
   }
}
```

6.12. forráskód. Szöveg elemzése

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.IO;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("Kérem adjon meg egy elvégezendő műveletet: ");
         string szoveg = Console.ReadLine();
         string elso_szam="";
         string masodik_szam = "";
         bool osszead = false;
         bool megvan_az_elso = false;
         for (int i = 0; i < szoveg.Length; i++)
         {
            if (Convert.ToString(szoveg[i]) != "+" &&
               Convert.ToString(szoveg[i]) != "-"
               && megvan_az_elso!=true)
            {
               elso_szam = elso_szam +
               Convert.ToString(szoveg[i]);
            }
            if (Convert.ToString(szoveg[i]) == "+")
            {
               osszead = true;
               megvan_az_elso = true;
            }
            if (Convert.ToString(szoveg[i]) == "-")
            {
               megvan_az_elso = true;
            }
            if (Convert.ToString(szoveg[i]) != "+" &&
               Convert.ToString(szoveg[i]) != "-"
                    && megvan_az_elso != false)
            {
               masodik_szam = masodik_szam +
                  Convert.ToString(szoveg[i]);
            }
         }
         if (osszead)
         {
            Console.Write("A két szám összege: {0}",
            int.Parse(elso_szam) + int.Parse(masodik_szam));
         }
         else Console.Write("A két szám különbsége: {0}",
            int.Parse(elso_szam) - int.Parse(masodik_szam));
         Console.ReadLine();
      }
   }
}
```

6.13. forráskód. Szöveg tisztítása

```
using System.Text;
using System.IO;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write ("Szöveg: ");
         string szoveg = Console.ReadLine();
         Console.Write ("Mit cserélünk: ");
         string cserelendo = Console.ReadLine();
         Console.Write ("Mire cseréljük: ");
         string ujszovegresz = Console.ReadLine();
         string ujteljesszoveg="";
         bool teljes_egyezes=true;
         int i = 0;
         while (i < szoveg.Length){
            if (Convert.ToString(szoveg[i]) !=
                     Convert.ToString(cserelendo[0])){
              ujteljesszoveg=ujteljesszoveg+szoveg[i];
            }
            if (Convert.ToString(szoveg[i]) ==
                      Convert.ToString(cserelendo[0])){
               for (int j = 0; j < cserelendo.Length; j++){
                  if (Convert.ToString(szoveg[j]) !=
                     Convert.ToString(cserelendo[j])){
                     teljes_egyezes=false;
                  }
                  teljes_egyezes = true;
               }
               if (!teljes_egyezes) ujteljesszoveg =
                              ujteljesszoveg + szoveg[i];
               if (teljes_egyezes){
                  ujteljesszoveg = ujteljesszoveg +
                  ujszovegresz;
                  i = i + (cserelendo.Length - 1);
               }
            }
            i++;
         }
         Console.WriteLine(ujteljesszoveg);
         Console.ReadLine();
```

```
      }
   }
}
```

6.14. forráskód. Nagy számok összeadása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ConsoleApplication1
{
   class Program{
      public static void Main(string[] args)
      {
         Console.Write("Kérem adjon meg egy számot: ");
         string szam_elso = Console.ReadLine();
         Console.Write("Kérem adjon meg még egy számot: ");
         string szam_masodik = Console.ReadLine();
         string eredmeny = "";
         int szam = 0 ;
         string csere="";
         if (szam_masodik.Length > szam_elso.Length)
         {
            csere = szam_elso;
            szam_elso = szam_masodik;
            szam_masodik = csere;
         }
         for (int i = 0; i < szam_masodik.Length; i++)
         {
            szam = int.Parse(Convert.ToString(
              szam_elso[szam_elso.Length - 1 - i])) +
                   int.Parse(Convert.ToString
                (szam_masodik[szam_masodik.Length - 1 - i]));
            eredmeny = eredmeny + Convert.ToString(szam);
         }
         for (int j = 0; j <
                  szam_elso.Length - szam_masodik.Length; j++)
         {
            eredmeny = eredmeny + szam_elso[szam_elso.Length -
                          szam_masodik.Length - j - 1];
         }
         Console.Write("Az eredmény: ");
         for (int k = 0; k < eredmeny.Length; k++)
         {
            Console.Write("{0}", eredmeny[eredmeny.Length-1-k]);
         }
         Console.ReadLine();
      }
   }
}
```

6.15. forráskód. Nagy számok szorzása

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ConsoleApplication1
{
   class Program
   {
      public static void Main(string[] args)
      {
         Console.Write("Kérem adjon meg egy számot: ");
         string szam_elso = Console.ReadLine();
         Console.Write("Kérem adjon meg még egy számot: ");
         string szam_masodik = Console.ReadLine();
         string nulla=""; string elsonulla = "";
         string eredmenyek = "";
         List<string> eredmeny = new List<string>();
         int szam = 0;

         for (int i = 0; i < szam_masodik.Length; i++){
            for (int j = 0; j < szam_elso.Length; j++){
               szam = int.Parse(Convert.ToString
                  (szam_elso[szam_elso.Length - 1 - j]))
                   * int.Parse(Convert.ToString(szam_masodik[i]));
               eredmenyek = eredmenyek + Convert.ToString(szam);
            }
            for (int k = 0; k < szam_elso.Length - 1 - i; k++){
               nulla = nulla + "0";
            }
            for (int t = 0; t < i; t++){
               elsonulla = elsonulla + "0";
            }
            eredmeny.Add(nulla + eredmenyek+elsonulla);
            eredmenyek = ""; nulla = ""; elsonulla = "";
         }
         string szamstring = "";
         int x, w,z;
         for (z = 1; z < eredmeny.Count; z++){
            for (w = 0; w < eredmeny[z].Length; w++){
               x = int.Parse(Convert.ToString(
                  eredmeny[z - 1][w])) +
                  int.Parse(Convert.ToString(eredmeny[z][w]));
                  szamstring = szamstring + Convert.ToString(x);
            }
```

```
                eredmeny[z] = szamstring;
                szamstring = "";
                Console.WriteLine("\n");
            }
            for (int q = 0; q < eredmeny[z - 1].Length; q++){
                Console.Write(eredmeny[z-1][eredmeny[z-1].Length-1-q]);
            }
            Console.ReadLine();
        }
    }
}
```

7. fejezet - A foreach ciklussal kapcsolatos feladatok (szerző: Király Roland)

Tartalom

A fejezet forráskódjai

> **7.1. feladat (Listák kezelése – *szint: 1*).** Készítsünk a 7.1 példaprogram alapján olyan alkalmazást, amely feltölt egy tetszőleges hosszúságú listát kétjegyű véletlen számokkal majd a lista elemeit kiírja a képernyőre.
>
> A lista feltöltését és a kiírását két külön ciklus végezze.

Magyarázat: A lista feltöréséhez használjunk for, vagy while ciklust, a kiíratáshoz viszont mindenképpen *foreach*-et.

A lista kezelés, és feltöltés egy ciklusban nem szerencsés dolog, mi se tegyük! A kétjegyű számok generálásához a véletlen szám generátor `Next()` metódusát kell paramétereznünk úgy, hogy 10-től 99-ig generáljon számokat,

> **7.2. feladat (Foreach elágazással – *szint: 1*).** A *foreach* vezérlő szerkezet nem csak egyszerűen a listák kiírására alkalmas. Segítségével bármilyen műveletet elvégezhetünk tetszőleges hosszúságú listák feldolgozása során. Ennek demonstrálására a 7.2 forrásszöveg alapján készítsünk egy feldolgozó rutint, ami egy lista páros elemeit írja ki a konzol képernyőre.

Magyarázat: A program elkészítése nagyon egyszerű, amennyiben meg tudjuk oldani a páros elemek kigyűjtését. A lista eleme akkor páros, ha kettővel osztva az osztás maradéka nulla. C# nyelvre fordítva ez a következőképpen néz ki: `lista[i] % 2 == 0`.

Figyelem! Sajnos ez a megoldás a *foreach* vezérlő szerkezet alkalmazása mellett nem működőképes, mivel nem hivatkozhatunk a lista elemeire az indexük alapján.

> **7.3. feladat (Foreach vagy For? – *szint: 1*).** Annak az eldöntésére, hogy mi a különbség a *for* ciklus és a *foreach* ciklus között, készítsük el egy tetszőleges tömb kiíratásának a programját mindkét változattal. A tömb elemei legyenek *int* típusúak és a felhasználótól kérjük be azokat. A feltöltés befejeztével írassuk ki a begyűjtött lista elemeit a képernyőre. A 7.3 programrészlet segít a megoldásban.

Magyarázat: A két különböző megoldásban a feltöltés nem különbözik egymástól, de a kiírás egészen máshogyan történik.

Míg a *for* ciklusban a tömb elemeire közvetlenül hivatkozhatunk a `listaneve[index]` formulával, pl.: egy t nevű tömb, és egy i nevű ciklusváltozó esetén a `WriteLine("{0}", t[i])` formában, a *foreach* esetében ez a következő képpen néz ki: `Console.WriteLine("{0}", x)`, ahol a *foreach* változója korábban az x nevet kapta.

> **7.4. feladat (Fibonacci számok – *szint: 3*).** Készítsük el a azt a programot, amely kiírja a képernyőre a Fibonacci számok közül az első néhányat. Egy lehetséges megoldást találunk a 7.4 példaprogramban. A kiíráshoz használjuk a *foreach* vezérlő szerkezetet, amely a korábban egy listában eltárolt Fibonacci számokat írja a képernyőre.

Magyarázat: A Fibonacci számokat egy tömb, vagy egy lista adatszerkezetben, konstansként tárolhatjuk az alábbi forrásban ismertetett módszerrel:

```
int[] fib = new int[]
        { 0, 1, 2, 3, 5, 8, 13 };
foreach (int f in fib)
{
    Console.WriteLine(f);
}
```

Információ: A Fibonacci számok a matematikában az egyik legismertebb rekurzív sorozat elemei. Az első két elem a nulla és az egy, a további elemeket minden esetben az előző két szám összegéből képezzük.

```
0,1,1,2,3,5,8,13,21,34,55,89,144,
```

A sorozatot először 1150-ben írta le két indiai matematikus, Gopala és Hemacsandra, akik a szanszkrit költészet elméleti kérdéseit vizsgálva ütköztek egy összegre bontási problémába (hányféleképpen lehet rövid és hosszú szótagokkal kitölteni egy adott időtartamot, ha egy hosszú szótag két rövidnek felel meg?).

Fibonacci 1202-ben újra felfedezte ezt a számsorozatot, majd felfedezését publikálta Liber Abaci „Könyv az abakuszról” című művében.

A könyvben ismertetett probléma, aminek a megoldását bemutatta az, hogy hogyan alakul a nyulak száma, ha feltételezzük, hogy

- az első hónapban csak egyetlen újszülött nyúl-pár van
- az újszülött nyúl-párok két hónap alatt válnak termékennyé
- minden termékeny nyúl-pár minden hónapban egy újabb párt szül
- valamint a nyulak örökké élnek…

Információ: Másik érdekesség a Fibonacci számokkal kapcsolatban, hogy nagy szerepük van Bartók Béla zeneműveiben.

Lendvai Ernő magyar zenetörténész Bartók Béla muzsikáját elemző könyvében mutatja be azt, hogyan tagolta zeneműveiben az egyes zenei gondolatok ütemsorrendjét a Fibonacci-szám hosszúságú szakaszok fölhasználásával Bartók.

A Lendvai Ernő által felfedezett Fibonacci szerkezet elméleti összefüggéseket Bartók Béla ösztönösen alkalmazta zenéjének formai arányrendszerében. forrás:

Wikipedia

A fejezet forráskódjai

7.1. forráskód. Listák kezelése

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         List<int> lista = new List<int>();
         Random rnd = new Random();
         for (int i = 0; i < 10; i++)
         {
            lista.Add(rnd.Next(10, 100));
         }
         foreach (int i in lista)
         {
               Console.Write("{0}, ", i);
         }
      }
   }
}
```

7.2. forráskód. Foreach elágazással

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         List<int> lista = new List<int>();
         Random rnd = new Random();
         for (int i = 0; i < 10; i++)
         {
            lista.Add(rnd.Next(10, 100));
         }
         Console.Write("A lista páros elemei: ");
         foreach (int i in lista)
         {
            if (i % 2 == 0)
            {
               Console.Write("{0}, ", i);
            }
         }
            Console.ReadLine();
      }
   }
}
```

7.3. forráskód. Fibonacchi számok

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         int[] fib = new int[]
            { 0, 1, 2, 3, 5, 8, 13 };
         Console.Write
           ("Fibonacci sorozat néhány tagja: ");
         foreach (int f in fib)
         {
            Console.Write("{0}, ",f);
         }
         Console.ReadLine();
      }
   }
}
```

8. fejezet - Ciklusok és vektorok használata összetett szöveg elemzésére (szerző: Király Roland)

Tartalom

A fejezet forráskódjai

8.1. feladat (Lexikális elemzés – *szint: 4*). Készítsük el egy egyszerű, determinisztikus, véges automata programját, amely automata a következőképpen írható le:

```
A terminális jelek = {0, 1, 2}.
Az állapotok = {q0, q1, q2},
Az állapot átmenetek  =
        (q0, 0, q1), (q0, 1, q1), (q0, 2, q2),
        (q1, 0, q1), (q1, 1, q2), (q1, 2, q0),
        (q2, 0, q2), (q2, 1, q0), (q2, 2, q1),
Kezdő állapot = {q0},
Elfogadó végállapot = {q0}
```

Magyarázat: Annak ellenére, hogy a program első látásra bonyolultnak tűnik, valójában egyszerűen megoldható.

A megoldáshoz használjunk *string* adattípust, melyben elhelyezzük a vizsgálandó szöveget. A szöveg elejétől haladva minden elemet meg kell vizsgálnunk.

A megoldás nem igényel komolyabb programozási ismereteket, de elkészítéséhez tisztában kell lenni a formális nyelvtanok és az automaták működésével.

Reguláris kifejezésekhez determinisztikus véges automata konstruálható. Az automata implementációja, állapotai, és az állapot átmenetek magas szintű programozási nyelveken jól implementálhatóak.

A véges automatának vannak belső állapotai, input *ABC*-je, állapot átmenetei, kezdő állapota és végállapota.

A szöveget, amelyre a vizsgálandó jeleket írták egyesével megvizsgálja és minden lépésben állapotot vált. Ha a szöveg végére érve elfogadó állapotba kerül, a szöveg jó, máskülönben hibás.

Ezt a műveletsort mutatja be a következő program, amely egyben a feladat megoldása is:

A program megírásához annyit kell tennünk, hogy az `STATE = ujallapot` rész helyére el kell helyeznünk egy elágazást, amely az egyes szövegelemekhez a megfelelő állapotokat rendeli (lásd: a 8.1, a 8.2, a 8.3 példaprogramokat).

Információ: A fentebb ismertetett automata valójában egy reguláris kifejezés automatája.

Reguláris kifejezéseket használunk akkor is, mikor az operációs rendszer parancssorában tevékenykedve például meg akarjuk keresni az összes szöveges fájlt a háttértáron `(*.txt)`.

8.2. feladat (Input handler – *szint: 2*). Készítsünk olyan alkalmazást, amely beolvas egy tetszőleges szöveget, majd eltávolítja belőle a szóközöket, valamint a sorvége jeleket (lásd: a 8.4, és a 8.5 példaprogramok)!

Magyarázat: A szöveget nem csak a billentyűzetről olvashatjuk be, hanem fájlból is.

Ha szeretnénk a szóközöket eltávolítani, vizsgáljuk meg a *string* típus *IndexOf* , és *Replace* metódusait.

Információ: Az input-handler a fordítóprogramok egyik részegysége. A forrásnyelvi szöveget beolvassa, majd az újsor és a kocsivissza karaktereket levágja a sorok végéről. Sok esetben ez a program a lexikális elemző részeként van implementálva. Az input-handler nem végez szintaktikai ellenőrzést, ezt a feladatot a lexikális elemző végzi, amely szintén része a fordító programoknak.

A fejezet forráskódjai

8.1. forráskód. Lexikális elemzés 1.

```
string állapot="a0";
string OK="OK";
int i=0;
while (i < szoveg.Length &&
    allapot !="error")
{
 STATE = "ujallapot"
 i++;
}
if (i < szoveg.Length)
{
 OK= "A hiba pozíciója"
     +i.ToString()+". "+ szoveg[i]
     +" nem eleme a nyelvnek";
}
```

8.2. forráskód. Lexikális elemzés 2.

```
switch (STATE + szoveg[i])
{
case "q00": STATE = "q0";break;
case "q01": STATE = "q1";break;
case "q02": STATE = "q2";break;
case "q10": STATE = "q1";break;
case "q11": STATE = "q2";break;
case "q12": STATE = "q0";break;
case "q20": STATE = "q2";break;
case "q21": STATE = "q0";break;
case "q22": STATE = "q1";break;
default: STATE = "error";break;
}
```

8.3. forráskód. Lexikális elemzés 3.

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
   class Program{
      static void Main(string[] args)
      {
         int[] terminalis_jelek = new int[3] { 0,1,2};
         string[] allapotok = new string[3] {"q0","q1","q1" };
         Program peldany = new Program();
         string [] szoveg=new string[11]
          {"q","0","1","q","1","1","q","2","1","q","0"};
         string lehetseges_kovetkezo_allapot = "";
         string kezdo_allapot = "q0"; string veg_allapot = "q0";
         string vizsgalando="";

         if (szoveg[0] + szoveg[1] == kezdo_allapot){
            int i = 0;
            while (i <= szoveg.Length - 5 &&
            lehetseges_kovetkezo_allapot != "hiba"){
               vizsgalando = szoveg[i] +
                      szoveg[i + 1] + szoveg[i + 2];
               lehetseges_kovetkezo_allapot =
               peldany.vizsgal(vizsgalando);
               i = i + 3;
            }

            if (i == szoveg.Length - 2){
               int szoveg_hossza = szoveg.Length;
               string vegallapot = szoveg[szoveg_hossza - 2]
                + szoveg[szoveg_hossza - 1];
               if (veg_allapot == vegallapot){
                  Console.WriteLine("Rendben.");
               }
            }
         }
         Console.ReadLine();
      }

      public string vizsgal(string a)
      {
         string kovetkezo_lehetseges_allapot;
         switch (a) {
            case "q00": kovetkezo_lehetseges_allapot = "q0"; break;
            case "q01": kovetkezo_lehetseges_allapot = "q1"; break;
            ...
            default: kovetkezo_lehetseges_allapot = "hiba"; break;
         }
         return kovetkezo_lehetseges_allapot;
      }
   }
}
```

8.4. forráskód. Forráskód kezelése 1.

```
string S="";
System.IO.StreamReader Stream = new
System.IO.StreamReader
(
      System.IO.File.OpenRead(source)
);

while (Stream.Peek() > -1)
{
   S += Stream.ReadLine();
}
...
```

8.5. forráskód. Forráskód kezelése 2.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         string szoveg = "";
         string sor = "";
         string ujsor = "";
         StreamReader beolvasott =
                new StreamReader("szoveg.txt");
         while (!beolvasott.EndOfStream)
         {
            sor = Convert.ToString(beolvasott.ReadLine());
            int i=0;
            while (i < sor.Length)
            {
               if (Convert.ToChar(sor[i]) != Convert.ToChar(" "))
               {
                  ujsor = ujsor + sor[i];
               }
               i++;
            }
            szoveg = szoveg + ujsor;
```

```
            ujsor = "";
          }
          Console.WriteLine(szoveg);
          Console.ReadLine();
        }
    }
}
```

9. fejezet - Mátrixok feltöltésével kapcsolatos feladatok (szerző: Hernyák Zoltán)

Tartalom

A fejezet forráskódjai

Az alábbi feladatokban egy egyszerű, kétdimenziós, egész számokból álló mátrixot fogunk feltölteni elemekkel. A különböző feltöltési feltételek és módszerek változatos feladatokká teszik ezt az alapvetően egyszerű tevékenységet.

9.1. feladat (Feltöltés billentyűzetről sorfolytonosan – *szint: 1*). Egy 5×4 méretű egész számokból álló mátrixot töltsünk fel sorfolytonosan billentyűzetről (először töltsük fel az első sorát, majd a második sorát, stb.)! A program összesen $5 \times 4 = 20$ értéket kérjen be. A megfogalmazásban az 5×4 méret csak példaképpen szerepel. A program a mátrix méretének változtatásával legyen képes eltérő méretekkel is működni! A program az adatbevitel végén jelenítse meg a mátrixban szereplő értékeket a képernyőn táblázatos formában!

Magyarázat: A mátrix méreteit (sor, oszlop) C# nyelven le lehet kérdezni a *.GetLength()* függvény segítségével, de kényelmesebb és elegánsabb megoldásnak tűnik, ha két konstanst definiálunk a sorok és oszlopok számához. A két konstans a program szövegében később helyettesítheti a konkrét méreteket:

9.1. forráskód. Konstansok használata

```
class Program
{
 const int N = 5;
 const int M = 4;
 public static void Main()
 {
   // matrix letrehozasa
   int[,] m = new int[N,M];
 }
}
```

Az adatbekérést a *Console.ReadLine* függvény segítségével kell megoldani, melynek string típusú értékét esetünkben szám típusúra kell konvertálni. Az adatbekérést dupla ciklusba kell helyezni.

9.2. forráskód. A mátrix bekérése

```
class Program
{
 const int N = 5;
 const int M = 4;
 public static void Main()
 {
   // matrix letrehozasa
   int[,] m = new int[N,M];
 }
}
```

A táblázatos megjelenítés során kalkuláljunk úgy, hogy a képernyőn 80 karakter jelenhet meg egy sorban, és 25 sorból áll a konzolablak! Feltételezzük, hogy a mátrixbeli számok sosem nagyobbak mint 999, így 3 karakteren elfér minden mátrixbeli szám. A *Console.Write* és *Console.WriteLine* képes formázottan kiírni számokat a képernyőre, pl. megadható, hogy a szám minimálisan hány karaktert foglaljon el a képernyőn. A szokásos *„{0}”* hivatkozást ki kell bővíteni a karakterek számával. A *„{0,4}”* alak azt jelenti, hogy a {0}. extra paramétert 4 karakter szélesen kell kiírni. Így a kiírás során az oszloposság biztosítható.

9.3. forráskód. Formázott kiírás

```
int x = 12;
Console.Write("{0,4}",x);
```

A teljes mátrixkiírás valahogy így néz ki (9.4. forráskód):

9.4. forráskód. Mátrix formázott kiírása

```
for(int i=0;i<N;i++)
{
 for(int j=0;j<M;j++)
 {
   Console.Write("{0,4}",t[i,j]);
 }
 Console.WriteLine();
}
```

9.2. feladat (Feltöltés billentyűzetről folyamatos kijelzéssel – *szint: 2*). A sorfolytonos feltöltés közben minden teljes sor bevitele után a mátrix már kitöltött elemeit táblázatos alakban jelenítsük meg a képernyőn! Tehát pl. a 3. sor bevitele után az első 3 sor jelenjen meg!

Magyarázat: Ez a feladat az előzőtől annyival bonyolultabb, hogy a bekérésbe beágyazva kell megoldani a kiírást. Ha a bekérés ciklusait *i* és *j* változónevekkel jelöltük, akkor a kiírás ciklusait nem jelölhetjük ugyanezekkel. A kiírást módosítani kell, hogy az *i*. sorig jelenítse csak meg a mátrixot (a 9.5. forráskódban ez a *k* cikluson múlik).

9.5. forráskód. Mátrix kiírása bekérés közben

```
for(int i=0;i<N;i++)
{
 // i. sor bekerese
 for(int j=0;j<M;j++)
 {
   Console.Write("Kerem {0},{1} elem erteket:",i,j);
   m[i,j] = int.Parse( Console.ReadLine() );
 }
 // kiiras az i. sorig
 Console.WriteLine("----------------------------");
 for(int k=0;k<=i;k++)
 {
   for(int l=0;l<M;l++)
   {
      Console.Write("{0,4}",t[k,l]);
   }
   Console.WriteLine();
 }
}
```

9.3. feladat (Feltöltés billentyűzetről – *szint: 1*). Egy 5×4 méretű egész számokból álló mátrixot töltsünk fel úgy billentyűzetről, hogy a felhasználó minden egyes adatbekérés során megadja sor- és oszlopkoordináták szerint, melyik mátrixelem értékét kívánja beírni, majd beírja magát a számértéket is! A program ügyeljen arra, hogy a mátrixelemek koordinátái a méreten belül maradjanak! A felhasználó a koordinátákat ne 0-tól, hanem 1-től számozva adhassa meg, vagyis $y \in [1 \dots 5]$, $x \in [1 \dots 4]$ értékekkel.

Magyarázat: Ez esetben nincs szükség egymásba ágyazott ciklusokra, mivel a bekérést egyszerűen 20-szor kell megismételni. Ügyeljünk arra, hogy 20 helyes bekérést kell végrehajtani, ezért ha hibás koordinátákat adnak meg, akkor azt nem számoljuk bele a 20-ba (a 9.27. forráskód). Ne feledjük el, hogy a mátrix sorai és oszlopai a forráskódban 0-tól számozódnak, így a bekért koordinátákból 1-et le kell vonni!

9.6. forráskód. Adatbekérés koordinátákkal

```
int db=0;
while (db<N*M)
{
 Console.Write("Kerem a matrix sorat [1..{0}]:",N);
 int i = int.Parse( Console.ReadLine() );
 Console.Write("Kerem a matrix oszlopat [1..{0}]:",M);
 int j = int.Parse( Console.ReadLine() );
 Console.Write("Kerem az erteket:");
 int x = int.Parse( Console.ReadLine() );
 if (1<=i && i<=N && 1<=j && j<=M)
 {
   m[i-1,j-1] = x;
   db++;
 }
 else Console.Write("Na ezt megegyszer, hibas volt.");
}
```

A probléma egy másik lehetséges kezelése, hogy amennyiben a felhasználó nem valós sor- és oszlopkoordinátát adna meg, úgy azt megismételtetjük vele, mindaddig, míg elfogadható értékeket kapunk. A bekérésnél mindig csak akkor lépünk a következő bekérésre, ha az előzőek már rendben voltak. Ekkor a tárolást megelőző *if* feltételvizsgálatra már nincs szükség. Valamint ekkor minden menetben sikeres adatbekérést hajtunk végre, így a külső 20-as ciklus átírható *for* ciklusra (lásd a 9.28. forráskód).

9.7. forráskód. Adatbekérés koordinátákkal v2.0

```
for (int db=0;db<<N*M;db++)
{
 // sor
 int i;
 do
 {
   Console.Write("Kerem a matrix sorat [1..{0}]:",N);
   i = int.Parse( Console.ReadLine() );
 }
 while (i<1 || i>N);
 // oszlop
 int j;
 do
 {
   Console.Write("Kerem a matrix oszlopat [1..{0}]:",M);
   j = int.Parse( Console.ReadLine() );
 }
 while (j<1 || j>M);
 // ertek
 Console.Write("Kerem az erteket:");
 int x = int.Parse( Console.ReadLine() );
 // tarolas
 m[i-1,j-1] = x;
}
```

Érdemes ez esetben függvényhívásokat szervezni az egyes részfeladatok megoldásának. Ez esetben a hibakijelzés is elegánsabban megoldható (lásd a 9.29. forráskód). A sor- és oszlopbekérő függvényt szándékosan két különböző megoldással készítettük el (9.29. forráskód).

9.8. forráskód. Adatbekérés függvények segítségével v3.0

```
static int sorBekeres()
{
int i;
Console.Write("Kerem a matrix sorat [1..{0}]:",N);
do
{
   i = int.Parse( Console.ReadLine() );
   if (i<1 || i>N) Console.WriteLine("Nem jo. Ujra!");
}
```

```
while (i<1 || i>N);
return i;
}
        //.............................................................
static int oszlopBekeres()
{
Console.Write("Kerem a matrix oszlopat [1..{0}]:",M);
while(true)
{
   int j = int.Parse( Console.ReadLine() );
   if (1<=j && j<=M) return j;
   else Console.WriteLine("Nem jo. Ujra!");
}
}
        //.............................................................
static void Main()
{
 ...
 for (int db=0;db<<N*M;db++)
 {
   int i=sorBekeres();
   int j=oszlopBekeres();
   Console.Write("Kerem az erteket:");
   int x = int.Parse( Console.ReadLine() );
   // tarolas
   m[i-1,j-1] = x;
 }
}
```

9.4. feladat (Egyszerre több érték listásan – *szint: 3*). A mátrix feltöltése közben lehessen egy időben több értéket is megadni! Ennek során a kezelő először megadja a cella koordinátáját, majd vesszővel elválasztva több számértéket. Ezt úgy kell kezelni, hogy a számértékek közül az első az adott x,y cellába kerüljön, a következő érték a sorfolytonosan következő cellába, stb. Ha több számérték került volna be a felsorolásba, mint amennyit az adott sor képes fogadni, úgy a felesleges értékeket hagyjuk figyelmen kívül.

Magyarázat: Ha egy időben több számot is be lehet írni, akkor a bevitel nem *int*-ben fogadható, hanem *string*-ben. A string ekkor vesszőkkel (vagy más határolójelekkel) elválasztva tartalmazza a számokat. A stringből legegyszerűbben a *split* függvénnyel lehet tömböt készíteni. A *split* (szétvág) használatakor meg kell adni a határolójelet, amely mentén a stringet elemekre kell bontani. A kapott stringtömbben az eredeti stringet alkotó listaelemek fognak szerepelni. Ezeket már egyesével a *parse* segítségével fel lehet dolgozni (a 9.30. forráskód).

9.9. forráskód. String felbontása a Split segítségével

```
// pl "12,24,35"
string s = Console.ReadLine();
string[] elemek = s.Split(',');
// ez esetben az elemek vektor 3 elemű lesz
// elemek[0] = "12"
// elemek[1] = "24"
// elemek[2] = "35"
```

9.5. feladat (Feltöltés folytatása I/N – *szint: 2*). A 9.3. feladatban megfogalmazott feltöltést módosítsuk annyiban, hogy a felhasználónak legyen lehetősége korábban is félbeszakítani! Ezt oldjuk meg úgy, hogy minden egyes mátrixelem beírása után jelenjen meg a *Szeretne újabb adatot beírni I/N* kérdés, melyre *N* válasz megadása esetén a bevitelből azonnal lépjen ki! A továbbiakban a program jelenítse meg a mátrixban szereplő értékeket a képernyőn táblázatos alakban!

Magyarázat: Az I/N választ bekérhetjük *Console.ReadLine()* segítségével is, de ekkor az I vagy N betű leütése után még *Enter*-t is kell ütni. Elegánsabb megoldás a *Console.ReadKey()* használata, mely ténylegesen egy billentyű leütésére vár, és eredményül egy *ConsoleKeyInfo* típusú értéket ad meg. Ez egy *rekord*, melynek különböző mezőiben szerepel nemcsak az, hogy melyik billentyűt ütötték le, de az is, hogy a módosítók (shift, control, alt) közül melyik volt eközben lenyomva (lásd a 9.31. forráskód).

9.10. forráskód. Console.ReadKey használata

```
ConsoleKeyInfo k = Console.ReadKey();
if (k.KeyChar == 'n' || k.KeyChar == 'N')
 Console.Write("NEM");
else if (k.KeyChar == 'i' || k.KeyChar == 'I')
 Console.Write("IGEN");
```

9.6. feladat (Módosítás ellenőrzése – *szint: 3*). A mátrix feltöltése közben előfordulhat, hogy a felhasználó egy olyan mátrixelem bevitelét végzi, amely már korábban kapott értéket. Ha ez előfordulna, úgy a koordináták beolvasása után írjuk ki a képernyőre, hogy *Figyelem: ez az elem már kapott értéket. Valóban módosítani akarja I/N?*! Ha a felhasználó I-t választ, úgy módosíthassa ezt az értéket, ellenkező esetben kérjünk újabb koordinátákat be!

Magyarázat: A mátrix minden egyes cellájában induláskor a típusának megfelelő *nulla* érték szerepel. Double típusú mátrixnál a 0.0, int típusú mátrixnál 0, karakter esetén a 0 kódú (`\0`) karakter, bool mátrix esetén a false, stb. Ezt ki tudjuk használni annak eldöntésére, hogy a mátrix adott cellája kapott-e már értéket korábban vagy sem.

Sokkal bonyolultabb a helyzet, ha a mátrixba maga a 0 mint érték is bekerülhet a felhasználói adatbevitel során. Ekkor nem lehet elkülöníteni, hogy egy mátrixcellának azért 0 az értéke, mert még nem írtak be oda semmit, vagy azért 0, mert ezt az értéket írták be. Ekkor még választhatjuk, hogy a mátrixot előfeltöltjük, és egy olyan számértéket helyezünk a cellákba, amely nem a 0, de valamiért tudjuk, hogy nem fordulhat elő adatbevitel közben. Ez az érték lehet akár a -100 is, de gyakoribb, hogy ilyen *közönséges* értéket nem könnyű találni. Ekkor a szélsőségesebb értékek tudnak segíteni, pl. a *int.MinValue* vagy *int.MaxValue*, mely konstansok az *int* típus esetén tárolható legkisebb és legnagyobb számértéket hordozzák.

9.11. forráskód. Mátrix előfeltöltése speciális értékekkel

```
for (int i = 0; i < N; i++)
 for (int j = 0; j < M; j++)
   t[i] = int.MinValue;
```

Előfordulhat azonban olyan eset is, amikor ezek a szélsőséges értékek sem eléggé speciálisak: nem tudjuk garantálni azt sem, hogy ezek nem szerepelnek a felhasználói inputban. Ekkor a mátrixunk egyedül alkalmatlan arra, hogy megkülönböztessük vele egymástól a definiált és értékekkel nem ellátott cellákat. Készítenünk kell egy második mátrixot is, amelynek celláiban azt tároljuk el, hogy az igazi mátrixunk melyik cellája kapott már értéket, melyik nem. Erre a logikai értékek tökéletesen megfelelnek, tehát ezen másodlagos mátrixunk alaptípusa lehet *bool*. A bool mátrixok cellái induláskor *false* értékűek, ezt az értéket tekintjük a *kapott-e már az igazi mátrixunk hasonló cellája értéket* kiinduló értékének is (egyelőre egyik cella sem kapott). Mivel ez tökéletes induló értéknek, a bool mátrix előfeltöltésére nincs szükség, az igazi *int*-es mátrixunk előfeltöltésére sincs (oda is megfelelnek az induláskori 0 értékek).

9.12. forráskód. Mátrix cellája kitöltött-e

```
class Program
{
 const int N = 5;
 const int M = 4;
 public static void Main()
 {
   // matrix letrehozasa
   int[,] m = new int[N,M];
   // cellkitoltott matrix letrehozasa
   bool[,] b = new bool[N,M];
 }
}
```

Amikor az *int*-es mátrixunk valamely [i,j] cellája értéket kap, ugyanakkor a bool mátrixunk [i,j] cellájába is kell rakni *true* értéket, jelezvén hogy ez a cella kitöltésre került. Ezek után már a bool mátrix alapján könnyű eldönteni, hogy az eredeti mátrix mely cellája volt kitöltve, és melyik nem volt.

9.7. feladat (Feltöltés befejezése 0-val – *szint: 2*). A korábban megfogalmazott *van még elem I/N* típusú befejezéssel az a gond, hogy legalább egy mátrixbeli elemet be kell írni, hogy befejezhessük a bevitelt. A felhasználónak ehelyett úgy kell jeleznie, hogy nincs további bevitel, hogy az x vagy y koordinátához 0 értéket ír be. Amennyiben az x-hez ír be 0-t, úgy az y értékét nem kell bekérni. A bevitel végén a mátrix jelenjen meg a képernyőn táblázatos alakban!

Magyarázat: Az eddigi feladatok után nincs jelentős probléma ezzel a feladattal. A ciklus kilépését az *x* vizsgálata után egy *break* segítségével oldhatjuk meg.

```
while (true)
{
 int x = int.Parse( Console.ReadLine() );
 if (x==0) break;
 ...
}
```

9.8. feladat (Nincs kitöltve jelzés – *szint: 3*). Amennyiben a felhasználó a bevitel végét kérné, a program ellenőrizze, hogy a mátrix minden eleme kitöltésre került-e! Amennyiben nem, úgy jelenítse meg a *még N kitöltetlen elem van, be kívánja fejezni I/N* üzenetet (*N* helyébe kerül a kitöltetlen elemek száma)! Amennyiben a felhasználó ennek ellenére kéri a bevitel befejezését, úgy a bevitel érjen véget! Nem válasz esetén a bevitel folytatódhasson! Ha a bevitel végét úgy kéri a felhasználó, hogy valóban már minden mátrixelem kitöltésre került, úgy a fenti üzenet ne jelenjen meg, a bevitel azonnal érjen véget! A program jelenítse meg a mátrixot a képernyőn táblázatos alakban!

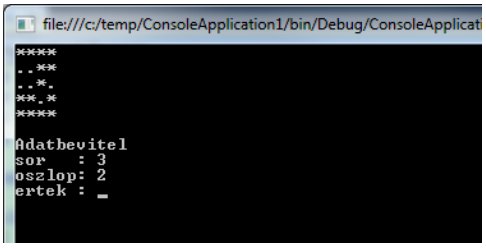
Magyarázat: Alkalmazni kell valamelyik módszert annak eldönthetőségére, hogy adott mátrixelem kitöltésre került-e vagy sem. Több módszert is megemlítettünk a 2.6. feladat kapcsán. A leguniverzálisabb módszer a mátrixszal egyező méretű *bool* mátrix alkalmazása. A kitöltetlen elemek számát ekkor a *bool* mátrixban (*segéd* mátrix) található *false* értékű elemek megszámolásával tudjuk kalkulálni (lásd a 9.34. forráskód).

9.13. forráskód. Kitöltetlen elemek száma

```
static int kitoltetlenElemekSzama()
{
  int db=0;
  for(int i=0;i<N;i++)
   for (int j=0;j<M;j++)
     if (seged[i,i]==false) db++;
  //
  return db;
}
```

9.9. feladat (Mátrix térképe – *szint: 2*). Oldjuk meg a bevitelt úgy, hogy a képernyő felső részén jelenjen meg a mátrix *térképe*, jelezvén, hol van kitöltött és kitöltetlen elem. Ezt úgy érjük el, hogy a kitöltött elemek helyén egy * (csillag) karakter jelenjen meg, a kitöltetlen elemek helyén pedig egy . (pont) karakter. Mivel így a felhasználó folyamatosan nyomon követheti, hol van feltöltött vagy feltöltetlen elem, a kilépés kérése esetén azonnal befejezhető a bevitel.

9.1. ábra. Mátrixtérkép, beviteli képernyő



Magyarázat: A mátrix térképét ki tudjuk rajzolni, ha alkalmazzuk például a 9.8. feladatban bemutatott logikai alapú mátrixot, melyben adminisztráljuk, melyik cella került kitöltésre, melyik nem.

9.10. feladat (Mátrix területi feltöltése – *szint: 2*). A mátrix feltöltése történjen billentyűzetről adatbevitellel az alábbi módon: a mátrixban elhatárolunk kis területeket, melyek a *nagy* mátrix kis egybefüggő téglalap alakú területei. Minden területnek van bal felső sarka (x,y) koordinátákkal megadva,

szélessége és magassága. A bevitel során kérjük be egy ilyen terület bal felső sarkának koordinátáját (x,y), majd a szélességét és magasságát, végül egy feltöltő X értéket! A mátrix adott részterületébe eső cellákat töltsük fel egységesen ezen X értékkel!

9.2. ábra. Mátrix területi feltöltése

31	24	37	17	56	27	47	47	20	40	39	23	18	40
15	56	10	23	17	16	53	23	38	17	43	16	39	25
16	43	12	12	12	12	12	12	12	10	39	22	44	44
34	16	12	12	12	12	12	12	12	28	56	17	12	48
52	52	12	12	12	12	12	12	12	18	17	56	16	18
25	23	12	12	12	12	12	12	12	10	24	30	53	53
38	53	36	33	22	56	17	17	19	35	54	45	31	12
13	47	43	58	31	59	39	42	40	11	50	16	18	34
23	24	50	52	31	40	16	23	46	43	13	36	45	39
54	52	13	13	15	21	29	25	10	10	47	26	12	15

Magyarázat: Ügyelni kell arra, hogy a felhasználó által megadott *x,y* koordináták, valamint a *szel,mag* szélesség, magasság értékek mellett nehogy a mátrixot alul-, vagy túlindexeljük. Ezt két módon előzhetjük meg. Az egyik módszer szerint az adatok bekérése után az értékeket „belőjük” a mátrixon belülire, levágjuk a terület esetleges kilógó részeit (9.35. forráskód). A másik módszer szerint minden egyes elemfeltöltés előtt ellenőrizzük annak sikerességét (9.36. forráskód).

9.14. forráskód. A terület elővizsgálata

```
int x   = int.Parse(Console.ReadLine());
int y   = int.Parse(Console.ReadLine());
int szel = int.Parse(Console.ReadLine());
int mag = int.Parse(Console.ReadLine());
int x   = int.Parse(Console.ReadLine());
// ---- ELOZETES KORREKCIO -----
// x negativ, kilog balra
if (x<0) { szel = szel+x; x=0;}
// x kilog jobbra
if (x>N-1) { szel=0; }
// y negativ, kilog fent
if (y<0) { mag = mag+y; x=0;}
// y kilog lent
if (y>=M) { mag=0; }
// x+szel kilog jobbra
if (x+szel>N-1) szel=N-x;
// y+mag kilog alul
if (y+mag>M-1) mag=M-y;
// ----- FELTOLTES ----
for(int i=x;i<x+szel;i++)
  for(int j=y;j+mag;j++)
   m[i,j]=x;
```

9.15. forráskód. A koordináták egyenkénti ellenőrzése

```
int x   = int.Parse(Console.ReadLine());
int y   = int.Parse(Console.ReadLine());
int szel = int.Parse(Console.ReadLine());
int mag = int.Parse(Console.ReadLine());
int x   = int.Parse(Console.ReadLine());
// ----- FELTOLTES ----
for(int i=x;i<x+szel;i++)
  for(int j=y;j+mag;j++)
   if (0<=i && i<N && 0<=j && j<M)
     m[i,j]=x;
```

A két módszer között az a különbség, hogy az első könnyebben elrontható, jól kell kalkulálni a manipulációkat, de utána gyors működésű ciklusokat kapunk. A második nehezen elrontható, de a hosszú ciklusok meneteiben minden egyes alkalommal feltételvizsgálatokat hajt végre, így a másikhoz képest mindenképpen lassabb. Ha azonban a feltöltendő területek koordinátái valóban billentyűzetes adatbevitelből származnak, akkor ez a lassúság nem észlelhető, hisz a felhasználó lassú gépelése lesz az igazi sebességbeli akadály.

9.11. feladat (Exceles bevitel – *szint: 3*). A mátrix elemeinek bevitelét oldjuk meg úgy, hogy induláskor megjelenítjük a képernyőn a mátrixot táblázatos alakban, a cellákban a 0 kezdőértékekkel! A felhasználónak legyen lehetősége a kurzornyilakkal kiválasztani, melyik cella értékét kívánja megadni az *Enter* leütésével! Ekkor a táblázat alján jelenjen meg a cella koordinátája, a cella jelenlegi értéke, és legyen lehetőség az új számérték beírására!

Magyarázat: A *Console.ReadKey* segítségével lehet megvalósítani a kurzorbillentyűs navigációt. Ekkor a kapott rekordban nem a *.KeyChar* mező az érdekes, hanem a *.Key*, amely egy felsorolás (enum) típusú érték. Ennek segítségével lehet eldönteni, mely vezérlőbillentyűt ütötték le (9.37. forráskód, 9.1. videó).

9.1. videó. Excel bevitel futási képernyője

9.16. forráskód. Navigálás a mátrixban

```
const int N = 5;
const int M = 4;
int[,] m = new int[N, M];

for (int k = 0; k < N; k++)
{
   for (int l = 0; l < M; l++)
      Console.Write(" {0,5} ", m[k, l]);
   Console.WriteLine();
}

int i = 0, j = 0;
bool folyt = true;
while (folyt)
{
   // m[i,j] cella kiemelese
   Console.SetCursorPosition(j * 7, i);
```

```
   Console.BackgroundColor = ConsoleColor.Green;
   Console.Write(" {0,5} ", m[i, j]);
   Console.BackgroundColor = ConsoleColor.Black;
   //
   ConsoleKeyInfo k = Console.ReadKey();
   // m[i,j] cella kiemeles megszuntetese
   Console.SetCursorPosition(j * 7,i );
   Console.BackgroundColor = ConsoleColor.Black;
   Console.Write(" {0,5} ", m[i, j]);
   // ..
   switch (k.Key)
   {
      case ConsoleKey.UpArrow:
         if (i > 0) i--;
         break;
      case ConsoleKey.DownArrow:
         if (i < N - 1) i++;
         break;
      case ConsoleKey.LeftArrow:
         if (j > 0) j--;
         break;
      case ConsoleKey.RightArrow:
         if (j < M - 1) j++;
         break;
      case ConsoleKey.Escape:
         folyt = false;
         break;
      case ConsoleKey.Enter:
         // m[i,j] bekerese
         // ...
         break;
   }
}
```

9.12. feladat (Feltöltés véletlen számokkal – *szint: 1*). Egy $N \times M$ méretű egész számokból álló mátrixot töltsünk fel véletlen számokkal, az $[A, B]$ intervallumból! A program induláskor kérje be az *N* és *M* értékét, valamint az *A* és *B* értékeket is! A program a feltöltés után jelenítse meg a képernyőn a kapott mátrixot táblázatos alakban!

Magyarázat: Csak egyetlen fontos hibalehetőség van. Tudnunk kell, hogy egyetlen *Random* példány tetszőlegesen sok véletlen szám előállítására is képes. Ugyanakkor az egymáshoz időben közel (például egy ciklus belsejében történő) *Random* példányok ugyanazon kezdőértékről indulnak, s ugyanazon véletlen számokat fogják előállítani. Ezért ügyeljünk arra, hogy a mátrix celláinak feltöltéséhez a ciklusok előtt deklarált egyetlen *Random* példányt használjunk (9.38. forráskód)!

9.17. forráskód. Feltöltés véletlen számokkal

```
Random rnd = new Random();
//
for (int i = 0; i < N; i++)
{
   for (int j = 0; j < M; j++)
      m[i, j] = rnd.Next(A,B+1);
}
```

9.13. feladat (Feltöltés fájlból – *szint: 4*). Egy $N \times M$ méretű egész számokból álló mátrixot töltsünk fel úgy, hogy a mátrixba kerülő értékek egy text fájlban vannak! A text fájl szerkezet szerinti első sora tartalmazza az N és M értékét szóközzel elválasztva, majd alatta N sor következik, mindegyikben M darab számérték, szintén szóközökkel elválasztva. A sorok végét a szokásos `\r \n` karakterek zárják. A program ügyeljen arra, hogy a text fájl hibás is lehet, vagyis egy sorban több vagy kevesebb mint M szám szerepelhet, és több vagy kevesebb mint N sora is lehet a text fájlnak! A beolvasás végén a program táblázatos formában jelenítse meg a képernyőn a beolvasott mátrixot, és jelezze, hogy tapasztalt-e hibát a text fájl feldolgozása közben!

Magyarázat: A fájlkezeléssel kapcsolatos osztályok egyrészt a *System.IO*, másrészt a *System.Text* névtérben vannak, ezért ezeket a névtereket is érdemes a kód elején a *using* segítségével beemelni. A fájl megnyitását a *StreamReader* osztály példányosításával lehet kezdeményezni. A konstruktor paramétere a fájl neve, valamint a kódlap, amellyel a fájl tartalma íródott. Az így készített *r* példányon keresztül lehet beolvasni egy sort a fájlból (*r.ReadLine()*, illetve le lehet ellenőrizni, hogy elértük-e már a fájl végét (*r.EndOfStream* property). A fájl adatainak kiolvasását követően a fájlt be kell zárni (*r.Close()* (vázlatosan a 9.39. forráskódban olvasható).

9.18. forráskód. Beolvasás fájlból

```
using System;
using System.Text;
using System.IO;

string fname = @"c:\adatok.txt";
StreamReader r = new StreamReader(fname, Encoding.Default);
while (!r.EndOfStream)
{
 string s = r.ReadLine();
 ...
}
r.Close();
```

A fájl egy sorának beolvasását követően a szóközzel határolt számokat tartalmazó stringet a korábban ismertetett módon a *Split* segítségével lehet feldarabolni, és a kinyert értékeket az *int.Parse* segítségével számmá alakítani, majd felhasználni (a feladatban kért hibakijelzés kezelése nélkül a 9.40. forráskódban olvasható).

9.19. forráskód. Beolvasás fájlból

```
StreamReader r = new StreamReader(fname, Encoding.Default);
// elso sor N es M erteke
string[] ee = r.ReadLine().Split(' ');
int N = int.Parse( ee[0] );
int M = int.Parse( ee[1] );
int[,] m = new int[N,M];
```

```
// a matrix sorainak beolvasas
int i=0;
while (!r.EndOfStream)
{
 string[] ss = r.ReadLine().Split(' ');
 for(int j=0;j<M;j++)
   m[i,j] = int.Parse(ss[j]);
 i++;
}
r.Close();
```

9.14. feladat (Labirintus beolvasása fájlból – *szint: 3*). Egy text fájlban * és . karakterekből (csillag és pont) álló sorok vannak. A sorok egyforma hosszúak, egyéb karaktert nem tartalmaznak. A * és . karakterek egy labirintust írnak le, ahol a * reprezentálja a falat, a . karakter a folyosót. Ezek együtt egy $N \times M$ méretű, karakterekből álló mátrixot írnak le, ahol N a sorok száma a text fájlban, M pedig a soronkénti karakterszám. A fájl első sorában egyetlen szám, a mátrix sorainak száma szerepel, a maradék sorokban a csillag és pont karakterekből álló rajzolat. Olvassuk be a mátrixot, majd jelenítsük meg a képernyőn oly módon, hogy a * karakterek pirossal, a . karakterek zöld színnel jelenjenek meg a képernyőn!

Magyarázat: A feladat olvasása során egy karakterekből álló mátrixot képzelünk el, melyben az egyes csillag és pont karaktereket el tudjuk tárolni. Természetesen a feladat megoldható ezen az alapon is. Képzeljük el azonban ehelyett azt, hogy egy *m* hosszú *s* string valójában egy *m* hosszú karakter típusú vektor! Ekkor ha van *n* darab ilyen hosszú stringünk, akkor van *n*×*m* karakterünk is.

Ezért sokat egyszerűsödhet a fájlbeolvasás, ha a beolvasott stringeket nem bontjuk fel karakterekre, hanem meghagyjuk azt eredeti string alakjukban. Ez igazából a későbbi feldolgozási műveleteket nem bonyolítja (a beolvasás a 9.41 kódban látható).

9.20. forráskód. Labirintus beolvasása fájlból

```
StreamReader r = new StreamReader(fname, Encoding.Default);
// elso sor N erteke
int N = int.Parse( Console.ReadLine() );
string[] m = new string[N];
for(int i=0;i<N;i++)
{
 m[i] = r.ReadLine();
}
r.Close();
```

A kijelzés is könnyen megoldható, minden egyes karakter kiírása előtt át kell váltani a megfelelő írási színre (a 9.42 forráskód). Vegyük észre, hogy ha *m* egy *string[]* stringek egy vektora, akkor a *m[i]* az *i*. stringet jelöli, a *m[i][j]* pedig az *i*. string *j*. karakterét. Ez most nem jelölhető *m[i,j]* módon!

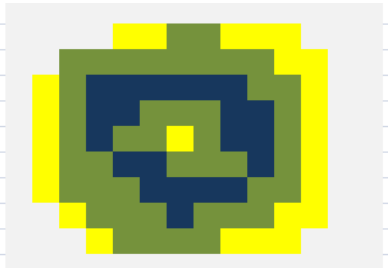
9.21. forráskód. Labirintus kiírása a képernyőre

```
for(int i=0;i<N;i++)
{
 for(int j=0;j<m[i].Length;j++)
 {
   if (m[i][j]=='.') Console.ForegroundColor = ConsoleColor.Green;
   else Console.ForegroundColor = ConsoleColor.Red;
   Console.Write(m[i][j]);
 }
 Console.WriteLine();
}
```

9.15. feladat (Sziget generálása – *szint: 4*). Az óceán közepén egy szabálytalan körvonalú sziget terül el, mely befoglalható egy $N \times N$ méretű téglalapba. A sziget közepén egy vulkán terül el. A téglalap minden négyzetkilométeréhez hozzárendelünk egy jellemző tengerszint feletti magasságértéket, melyet méréssel és átlagolással határoztunk meg. Generáljunk egy 20×20 méretű mátrixot, amely lehetne akár ezen sziget magassági értékeit tároló mátrixa is! A magasságértékek $[0 \dots 500]$ közötti (méterben értett) értékek legyenek! Jelenítsük meg a mátrixot a képernyőn táblázatos alakban, használjunk színeket is a különböző magasságértékek esetén sávosan (pl: $[0 \dots 100]$ legyen sárga, $[100 \dots 200]$ legyen zöld, stb.)! A 9.3. ábrán látható egy minta, ahol a sötétebb színek a magasabb részeket jelölik, a világosabbak pedig az alacsonyabbakat. Ügyeljünk a következőkre:

- a sziget a közepe fele haladva magasodik, tehát minél *beljebb* vagyunk, annál valószínűbben szerepeljenek nagyobb számértékek a mátrixban,
- a sziget közepe tájékán elterülő vulkán krátere jóval alacsonyabb, mint a kráter szélei, ez egy cellát jelent a mátrix esetén (magassága [200,300] közé esik).

9.3. ábra. A sziget egy lehetséges kinézete



Magyarázat: A sziget generálását érdemes a közepén kezdeni, a vulkán kráterével. Generáljunk oda egy kisebb értéket, a kráter közepének alacsony szintjét jelölve! A vulkán kráterének pozícióját oly módon határozhatjuk meg, hogy meghatározzuk a szélesség/2, magasság/2 pozíciót (a mátrix kellős közepe), majd ehhez véletlen $[-1 \dots 1]$ értéket adunk x és y irányban is. Ily módon a kiinduló pozíciónk ugyan középre esik, de mégsem pontosan középre (tároljuk el ezt a pozíciót *x0* és *y0* változókba).

A következő lépés, hogy az *x0* és *y0* pozíciót „hízlaljuk” *n* vastagsággal. Ezt több módon is meg lehet tenni. Az egyik legegyszerűbb módszer, hogy szkenneljük a mátrixot soronként (és oszloponként), keresünk benne olyan cellát, amely egyelőre kitöltetlen, de a vele szomszédos cella *szimpatikus*. Esetünkben a szimpatikus cella

az *x0*, *y0* (8 ilyen cellát találunk, lásd a 9.4. ábra). Hogy ne kerüljön minden ilyen cella kiválasztásra, minden cella kiválasztásának esélyét csökkentsük le 100%-ról mondjuk 70%-ra! Ekkor bizonyos szomszédos cellák kiválasztásra kerülnek, mások nem (9.5. ábra).

9.4. ábra. A vulkán krátere és a 8 szomszéd

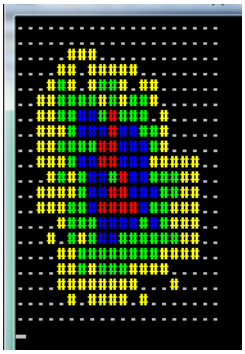
9.5. ábra. A vulkán krátere és a véletlenszerű kiválasztás

A következő fázisban vonjuk be a *szimpatikus* cellák körébe az előző körben kiválasztott és átszínezett cellákat is! Igazából fogalmazhatunk úgy is, hogy minden cella *szimpatikus*, amely ki van töltve. Az előző módszert újra alkalmazva megint válasszuk ki a *szimaptikus* cellák szomszédjait 70% eséllyel! Ekkor egy vastagabb kitöltést kapunk (9.6. ábra). Jegyezzük meg, hogy a második menet kiválasztási esélyét akár csökkenthetjük is, hogy a „vastagodás” ne legyen olyan látványos! Valamint jegyezzük meg, hogy a második menetben kiválasztott cellák maximum kettő távolságra lehetnek a kezdeti pozíciótól!

9.6. ábra. A vulkán krátere kétmenetes vastagítás után

A további körökben is ugyanezt fogjuk tenni, minden menetben legfeljebb egytávolságnyival „vastagítva” a kiinduló pontunkat. A nem 100% eséllyel történő kiválasztás során egy nem teljesen szabályos körvonalú befestett területet fogunk kialakítani a mátrixban. Az egyes menetekben más-más „szín” használva a kiválasztott cellák befestéséhez, kialakíthatjuk a réteges vastagítást is. A „szín” esetén itt most használhatunk más-más számintervallumot, amelyből a véletlen számokat generáljuk. A kitöltéshez használt forráskódok az 9.43 ... 9.45. forráskódokban, a futási eredmény egy lehetséges képernyőképe a 9.7. ábrán látható.

9.7. ábra. A véletlen vastagítás eredménye



9.22. forráskód. A vastagítás kódja (1. rész)

```
const int N = 20;
static int[,] m = new int[N, N];
static Random rnd = new Random();
//
static void vulkanKratere(int a, int f)
{
   int x = (N / 2) + rnd.Next(-1,+2);
   int y = (N / 2) + rnd.Next(-1, +2);
   m[x, y] = rnd.Next(a, f + 1);
}
```

9.23. forráskód. A vastagítás kódja (2. rész)

```
static bool szomszedSzimpatikus(int x, int y)
{
   for (int i = x - 1; i <= x + 1; i++)
      for (int j = y - 1; j <= y + 1; j++)
         if (0 <= i && i < N && 0 <= j && j < N && m[i, j] > 0)
            return true;
   //
   return false;
}
         //.............................................................
static void vastagit(int a, int f, int esely)
{
   int[,] temp = (int[,])(m.Clone());
   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
         if (temp[i,j]==0 && szomszedSzimpatikus(i, j))
            if (rnd.Next(0,100)<esely)
               temp[i, j] = rnd.Next(a, f + 1);
```

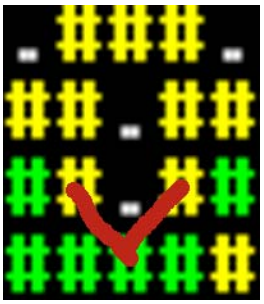
```
   m = temp;
}
```

9.24. forráskód. A vastagítás kódja (3. rész)

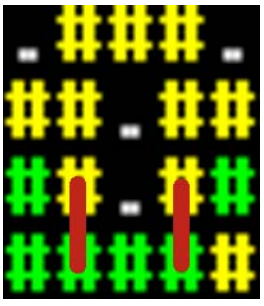
```
static void Main()
{
   vulkanKratere(200, 300);
   for(int i = 0;i<2;i++)
      vastagit(400, 500, 70-i*10);
   for (int i = 0; i < 2; i++)
      vastagit(300, 400, 70 - i * 10);
   for (int i = 0; i < 2; i++)
      vastagit(200, 300, 70 - i * 10);
   for (int i = 0; i < 2; i++)
      vastagit(100, 200, 70 - i * 10);

   //
   for (int i = 0; i < N; i++)
   {
      for (int j = 0; j < N; j++)
      {
         ConsoleColor c = ConsoleColor.Gray;
         if (m[i, j] < 100) c = ConsoleColor.Gray;
         else if (m[i, j] < 200) c = ConsoleColor.Yellow;
         else if (m[i, j] < 300) c = ConsoleColor.Green;
         else if (m[i, j] < 400) c = ConsoleColor.Blue;
         else if (m[i, j] < 500) c = ConsoleColor.Red;
         Console.ForegroundColor = c;
         if (m[i, j] == 0) Console.Write('.');
         else Console.Write('#');
      }
      Console.WriteLine();
   }
   Console.ReadKey();
}
```

Ha alaposan megfigyeljük a futási eredményt, láthatjuk, hogy már majdnem készen vagyunk. Van azonban egy ijesztően nagy gond. A kifestett területek „belsejében” tengerszintmagasságokat észlelhetünk. Ez nem csak annak egyenes következménye, hogy a vastagítás során mind a 8 irányban ellenőrizzük a *szimpatikus* cella szomszédságát (9.8. ábra), ez akkor is kialakulhat, ha csak 4 irányban ellenőrizzük ezeket (9.9. ábra). A 8 irány miatt könnyű szomszédot találni, miközben a 2 cella közöttinek is van szomszédja, de az esély elvétése miatt egy cella kimarad.

9.8. ábra. A 8 irányú ellenőrzés miatt kimarad 1 cella

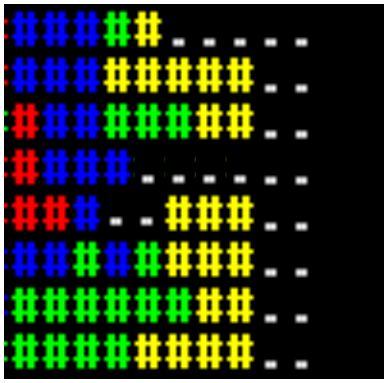


9.9. ábra. A 4 irányú ellenőrzéskor is kimaradhat 1 cella



Sajnos ezen egyszerűen azonban nem tudunk segíteni, mivel ez a sziget szélén normális jelenség, sőt, akkor is, ha a cella nem a szélén van, de esetleg egy szigetről a tengerbe ömlő folyó deltája, s ily módon mélyen bemetsz a sziget belsejébe (9.10. ábra).

9.10. ábra. A szélről induló benyúlás nem hibás

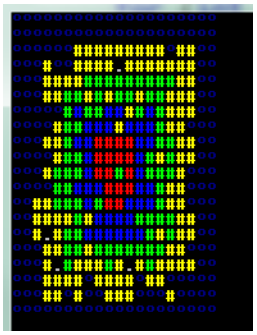


Ebből egy fontos dolog következik. A generálás (vastagítás) közben ezek a szünetek nem szűrhetőek ki, nem korrigálhatóak, mivel akár a végső állapotban is benne maradhatnak. Helyette egy utólagos szűrési és korrigálási fázisra van szükség, hogy a belső kis tócsákat megszüntessük. Persze, ha megengedett a sziget belsejében a tengerszint magasságában lévő területek jelenléte (beszüremlő tengervízi tavak, alacsonyan fekvő lapos földrészek, stb.), akkor ezen korrigálási lépésre nincs is szükség.

Tegyük fel, hogy nem megengedett! Hogyan különítsük el a szigetet körbefonó víz 0 magasságú celláit a hibásnak számító belső 0 értékű cellákról? A kulcs: a belső részek körbe vannak zárva magasabb részekkel. Itt megint lehet vitázni, hogy a bezárást mind a 8 irányból igényeljük-e, vagy 4 irányú bezárást már elégnek tekintünk. De ez csak a detektáló algoritmus apró módosításán múlik. A feladat egyelőre úgy fogalmazható meg: különítsük el a belső, bezárt 0 magasságú cellákat a külső (normális) 0 magasságú celláktól.

A megoldást a festő algoritmus jelentheti. Feltételezhetjük, hogy a 0,0 pozíción tengervíz van (ha ezt nem feltételezhetjük, akkor keresnünk kell egy szélső cellát, ahol 0 magasság van – az biztosan tengervíznek tekinthető). Ha ilyet nem találunk, akkor készen is vagyunk: minden 0 magasságú cella biztosan hibás! Ezt tehát egyelőre zárjuk ki, legyen egy kiinduló x_0, y_0 cellánk, ahol biztosan tengervíz van. Ezen cellából kiindulva *fessük be* a tenger vizét (4 vagy 8 irányban lépkedve, a korábban említettektől függően)! Ezek után amely 0 magasságú cellához nem jutott el a festés, az „belső cella”, vagyis hibásnak tekinthető a benne szereplő 0 érték. A 9.11. ábrán látható a 4 irányú festés eredménye, a tengervizet sötétkék színű pöttyök szimbolizálják, míg a belső, „hibás” cellák maradtak pont karakterek. A festő algoritmus a 9.46. forráskódban olvasható, indítása a külső *jolBefest();* függvény hívásával történik meg. Ez minden szélső cellában talált 0 (tengervíz) cellából kiindulva elvégzi a festést.

9.11. ábra. Hibákat tartalmazó sziget



9.25. forráskód. Festő algoritmus

```
static void befest(int x, int y)
{
   if (m[x, y] != 0) return;
   m[x, y] = 1;
   if (x > 0) befest(x - 1, y);
   if (x <N-1) befest(x + 1, y);
   if (y > 0) befest(x, y-1);
   if (y < N - 1) befest(x, y+1);
}
         //.............................................................
static void jolBefest()
{
   for (int i = 0; i < N; i++)
   {
      if (m[i, 0] == 0) befest(i, 0);
      if (m[0, i] == 0) befest(0, i);
      if (m[i, N-1] == 0) befest(i,N-1);
      if (m[N - 1,i] == 0) befest(N - 1,i);
   }
}
```

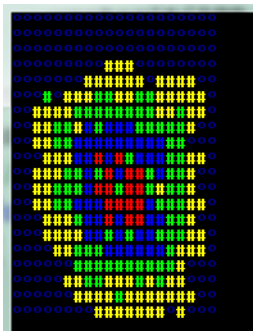
Hogyan korrigáljuk utólag a 0 magasságú hibás cellát? Vegyük a szomszédait, amelyek nem 0 értékűek, és állítsuk be a hibás cella magasságát egy átlagértékre (esetleg kis véletlen értéket hozzáadva, mondjuk ± 20 értékben, persze ügyelve, nehogy negatívba csússzunk át, vagy átlépjük az 500-as maximumot! A korrekciót végző függvények (a festés követően indíthatóak) a 9.47. forráskódban olvashatóak, a végeredmény a 9.12. ábrán látható.

9.26. forráskód. Korrekciózás a belső cellákra

```
static int atlagSzamit(int x, int y)
{
   int ossz = 0, db = 0;
   for (int i = x - 1; i <= x + 1; i++)
      for (int j = y - 1; j <= y + 1; j++)
         if (0 <= i && i < N && 0 <= j && j < N && m[i, j] > 0)
         {
            ossz += m[i];
```

```
            db++;
         }
   if (db == 0) return 0;
   else return ossz/db;
}
         //.............................................................
static void korrekcio()
{
   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
         if (m[i, j] == 0)
         {
            int x = atlagSzamit(i, j)+rnd.Next(-20,20);
            if (x<0) x=0;
            else if (x>500) x=500;
            m[i, j] = x;
         }
}
```

9.12. ábra. A korrekciózott, garantáltan „tökéletes” eredmény

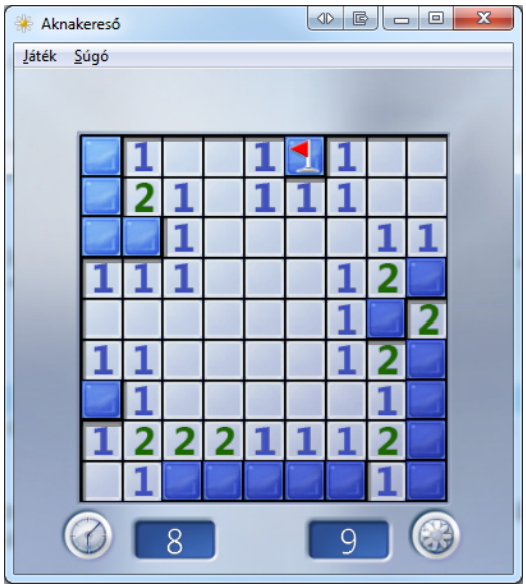


9.16. feladat (Aknakereső – *szint: 2*). Az ismert *aknakereső* játékban egy $N \times M$ méretű mátrixba kell *K* darab *aknát* elhelyezni. Az aknák elhelyezésének szabályai:

- ne tegyünk kétszer ugyanarra a helyre aknát, hogy a *K* darab akna a területen fellelhető legyen *K* különböző cellában,
- az aknák oldalukkal, sarkukkal szomszédos cellákba is kerülhetnek,
- az aknák a mátrix szélső celláiba is kerülhetnek.

A megfelelően feltöltött mátrixot jelenítsük meg a képernyőn oly módon, hogy az üres cellákat zöld színű . (pont) karakterek szimbolizálják, míg az aknát tartalmazó cellákat piros színű * (csillag) karakterek jelezzék!

9.13. ábra. Az aknakereső képernyője grafikusan



Magyarázat: Ez egy rendkívül könnyű feladat, mivel csak arra kell ügyelni, hogy ha olyan *x*, *y* koordinátát generálnánk, ahova már helyeztünk aknát korábban, akkor új koordinátapárt kell generálnunk. Akkor végeztünk, ha *K* üres cella koordinátáját sikerült generálnunk. A kiírás az előző feladatban leírtak szerint szintén egyszerű művelet.

Sokkal izgalmasabb feladat az aknakereső játék azon része, amikor a kész, aknákkal teli mátrix valamely *x*, *y* koordinátájú pozícióját a játékos kiválasztja. Ha ott akna van – a játék véget ér. Ha nincs ott akna, akkor ezen koordinátából induló *befestéssel* meghatározhatjuk, mely terület tárul fel a játéktérből. A festő algoritmus szintén az előző feladat megoldása során került bemutatásra.

9.17. feladat (Torpedó játék – *szint: 5*). Egy 20×20 méretű területen *hajókat* helyezünk el. A hajók mérete és száma különböző:

- *csónakok* (egyetlen cellát foglalnak el),

- *halászhajók* (két szomszédos cellát foglalnak el),
- *cirkálók* (három cella méretűek),
- *rombolók* (négy cella méretűek),
- *anyahajók* (öt cella méretűek).

A hajók alakja tetszőleges lehet, feltéve ha összefüggőek. Összefüggő akkor egy hajó, ha a hajót alkotó minden cella legalább egy másik cellával él mentén érintkezik.

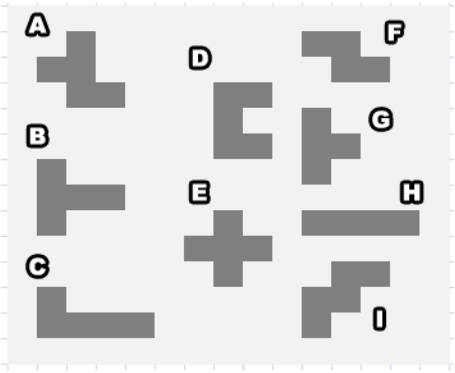
Készítsünk olyan programot, amely egy 20×20 méretű mátrixban elkészíti a hajók egy véletlenszerű elhelyezését oly módon, hogy 1 db anyahajó, 2 db hadihajó, 3 db cirkáló, 4 halászhajó és 5 csónak kerül elhelyezésre! A hajók egymással nem érintkezhetnek, még a sarkaik mentén sem, vagyis két különböző hajót alkotó cella még sarkával sem érhet össze. A hajók viszont a terület széleit elérhetik.

A programot úgy kell elkészíteni, hogy rugalmasan alkalmazkodjon a hajók darabszámának változásához, vagyis a hajók számát egy konstans formájában adjuk meg. A konstans értékének átírásakor a program megfelelő számú hajót helyezzen el! A program ügyeljen arra, hogy nagy mennyiségű hajó egyszerűen nem helyezhető el a területen (mivel a hajók nem érhetnek össze)! A program ilyen esetben se kerüljön végtelen ciklusba, jelezze ki a megoldhatatlanságot, és álljon le a futása!

A program a sikeres elhelyezés végén jelenítse meg a hajókat a képernyőn, * karakterrel jelezve a hajót felépítő cellákat, és . karakterrel a vizet!

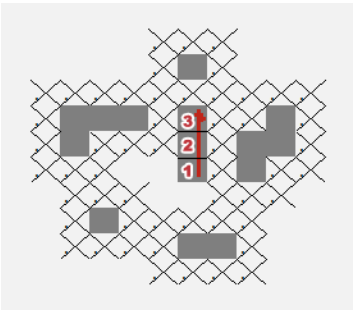
Magyarázat: Ez egy nagyon nehéz feladat, ha jó minőségű megoldást akarunk készíteni. Először is vegyük a hajók alakját! A 9.14. ábrán látható alakú hajók megjelenése (is) kívánatos. Ez azért fontos, mert az első „nehéz” dolog olyan algoritmust írni, amely ilyen alakú hajókat is képes generálni. Egyszerűbb olyanban gondolkodni, ahol egy kiinduló *x,y* koordinátából lineárisan növesztjük a hajó alakját (vagyis minden új cellát az előző cellához kapcsolunk a 4 irány egyikében). Ekkor például az *A*, *B*, *E* alakú hajók nem kerülnének generálásra.

9.14. ábra. A nagyobb hajók alakjainak néhány variációja



Egyébként is fontos a tetszőleges alakú hajók generálásának képessége, mivel egy rossz kezdőpozícióból rossz irányban indított hajógenerálás akár zsákutcába is vezethetne, ami a jó minőségű megoldásba nem fér bele. A 9.15. ábrán látható szituációban a kezdőpozíciónk (1) egy olyan területre mutat, amelyet a korábban generált hajók már erősen körülzártak. A felfele indulás után a (3)-as pozíciótól kezdve a lineáris generálás zsákutcába jutna, míg az ügyesebb generáló algoritmus képes lenne befejezni az 5-ös méretű anyahajó generálását ezek után is (az ábrán a sötétszürke mezők a hajók, a keresztben áthúzott cellák a hajók környezetét jelölik).

9.15. ábra. A lineáris irányú haladás bajba juthat



A tetszőleges alakú hajó generálása működhet úgy is, hogy a hajó elemeinek generálása során egy listába gyűjtjük az elemeinek koordinátáit (kezdetben a legelső eleme kerül bele, melynek pozíciókiválasztásáról később lesz szó). Amikor új elemet kívánunk még a hajóhoz csatolni, akkor ezen listából véletlenszerűen kiválasztunk egy már létező cellát, és egy véletlenszerű folytatási irányt (észak/dél/kelet/nyugat), majd megpróbáljuk ebbe az irányba folytatni a hajót. Ha nem sikerül, akkor választhatunk másik létező cellát és/vagy másik irányt. Ha már minden lerakott cellát és a belőlük kiinduló minden irányt kipróbálta az algoritmus – akkor beláthatjuk, hogy az adott kezdőpozícióból kiindulva nem lehet ezt a (nagyméretű) hajót lerakni.

Ehhez támogatásképpen kihasználhatjuk azt, hogy a listából törölni is lehet elemet. Tegyük fel, hogy már három cellát leraktunk a generálandó 5 méretű anyahajóból. A lista ekkor mindhárom cella mind a négy irányú folytatásának koordinátáit tartalmazza, 12 elemet. Véletlenszerűen választunk a 12 elemből, és ellenőrizzük a célt, hogy oda valóban letehetjük-e a folytatást. Ha sikerül, akkor 4 cellánk van, és újrageneráljuk a listát immár 16 elemmel. Ha nem sikerül, akkor a hibás próbálkozás koordinátáit töröljük a listából (1-gyel kevesebb elem marad). Ha a lista elfogy, akkor a hajót nem lehet folytatni, a kiindulási pontból a hajót nem lehet befejezni.

Ha egy hajót a mátrix egyetlen kezdőpozíciójából sem lehet már lerakni, akkor nagy a baj. Nem feltétlenül végzetes, de nagy. Ekkor az utolsó sikeresen lerakott hajót törölnünk kell, majd megpróbálni lerakni máshova vagy más alakkal, és újra kísérletet tenni a következő hajó lerakására is.

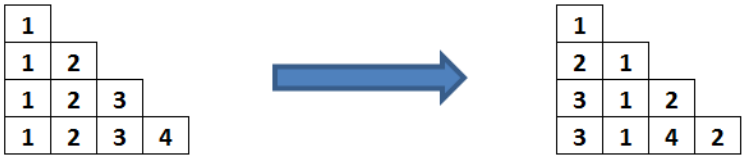
Érezhető, hogy itt a visszalépéses keresésre (*backtrack*) lesz szükség. A hajók lerakásának kezdőpozícióját véletlenszerűen bár, de szervezett sorrendben kell kiválasztani. A visszalépés során ugyanis másik pozíciót kell keresni, olyat, amely még nem került kiválasztásra. Ehhez a véletlen *x,y* koordinátaválasztás nem eléggé kifinomult. Helyette javasolt az $N \times M$ koordináta listába szervezése, majd véletlen összekeverése. A listát sorban haladva dolgozhatjuk fel, a kapott koordináták mégis véletlenszerű sorrendet produkálnak. Így megoldhatjuk, hogy a visszalépés során új kezdőpozíciót választhassunk a hajóknak, módszeresen kipróbálhassuk az összes kezdőpozíciót a véletlenszerűség megőrzése mellett.

A következő probléma az adott pozícióból kiinduló összes alakzat generálása. Nem arról van most szó, hogy ha nem rakható le a hajó, akkor bizonyosodjunk meg róla, hanem arról, hogy adott pozícióból kiindulva többféle alakban is lerakható.

Elképzelhető olyan szituáció, hogy az *N*. hajó lerakása adott kezdőpozícióból lehetetlen, de ha visszalépünk az $N-1$. (sikeresen lerakott) hajóhoz, és őt egy másik alakzat formájában rakjuk le, akkor az *N*. hajót a korábban sikertelen kezdőpozícióról egy újabb próbálkozás során már sikeresen le lehet rakni.

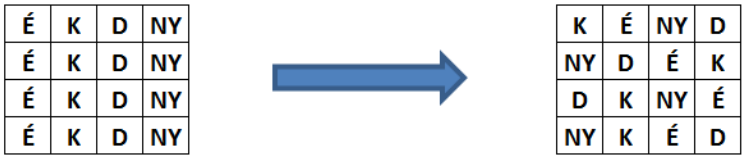
Az első probléma ez ügyben: hogyan tudunk módszeresen, de véletlenszerűen építkezni a már meglévő cellákból kiindulva? Az *n* cellából álló hajónk első lépésben csak 1 cellából áll, a kezdőpozíción. Ha sikerül még egy cellát leraknunk a környezetében, akkor már két cellánk lesz, és a harmadik cella helyének keresésekor kiindulhatunk az elsőből és a másodikból is. Készítsünk egy tervet előre! Generáljuk le a 9.16. ábrán látható háromszögmátrixot, soronként permutálva a szabályosan felépített bal oldali tervet! Ennek megfelelően a második cella generálásához az 1-es cellát vesszük kiindulási alapként (első sor, mondjuk itt más lehetőség nincs is), a harmadik cella helyét először a 2-esből, ha onnan nem járunk sikerrel, akkor az 1-esből kezdjük el (második sor). A negyedik cella helyének generálását elsőként a 3-asból, ha onnan nem megy, akkor az 1-esből, ha onnan sem, akkor a 2-esből fogjuk megpróbálni (3-as sor), stb.

9.16. ábra. A cellaválasztás véletlenszerűsítése



Ha megvan, melyik cellából indulunk ki, akkor az elhelyezést kell véletlenszerű, de módszeres módon megoldani. Mivel csak 4 irányba tudunk szomszédos cellát választani a fejlesztés során, jelöljük el ezeket É, D, K, NY módon (vagy 0, 1, 2, 3 számokkal)! Egy ötcellás hajó esetén négyszer kell továbblépési irányt választani valamely már elhelyezett cellából, ezért készítsük el a 9.17. ábrán látható iránymátrixot először szabályosan felépítve, majd soronként, cserék segítségével permutálva. Ekkor például a harmadik cella helyét úgy határozzuk meg, hogy az előzőekben kiválasztott, már lerakott cellából (első vagy második cella) először nyugatra próbálkozunk. Ha oda nem lehet valamiért, akkor dél, majd észak, végül kelet felé próbáljuk meg (az iránymátrix második sora). Ha egyik sem válik be, akkor másik cellából próbálkozunk a továbbfejlesztéssel. Ha semelyik (sem az első, sem a második) korábban lerakott cellából nem lehet a harmadik cellát lerakni, akkor felszedjük a második cellát, és azt próbáljuk meg új helyre lerakni.

9.17. ábra. Az irányok összekeverése



A visszalépéses keresés révén ekkor a hajó következő lerakandó celláját véletlenszerű irányokban, a következő cellát véletlenszerű (korábban már lerakott) cellából kiindulva próbáljuk meg lerakni. Ha sikerül, akkor lépünk egyet előre, és megpróbáljuk a hajó újabb celláját lerakni. Ha sikerül, akkor a hajó teljes egészében felépül. Ha valamely lerakási mintából – melynek során *m* cellát már sikeresen letettünk – egyáltalán nem sikerül a hajó újabb cellájának elhelyezése, akkor visszalépünk *m – 1* sikeres lerakásig, és az *m*. cellát máshova próbáljuk lerakni. Ha a visszalépés során az 1. cellát is fel kell szednünk, akkor a hajó lerakása adott kezdőpozíciótól lehetetlen, semmilyen alakban sem kivitelezhető. Ekkor visszalépünk az előző sikeresen lerakott hajóhoz, és ezt próbáljuk meg más alakban, de egyelőre ugyanazon kezdőpozícióból letenni. Ha ezt a hajót ebből a kezdőpozícióból már semmilyen más alakban sem tudjuk lerakni, akkor visszalépünk az őt megelőző hajóhoz. Ha a visszalépés során az első hajót kell más pozícióba rakni, akkor azt is meg kell tennünk. Ha az első hajót már minden létező pozícióból megpróbáltuk letenni, de a továbbiakban sosem sikerült az összes hajót lerakni, akkor a hajók elhelyezése az adott méretű pályán lehetetlen.

A fejezet forráskódjai

9.27. forráskód. Adatbekérés koordinátákkal

```
int db=0;
while (db<N*M)
{
 Console.Write("Kerem a matrix sorat [1..{0}]:",N);
 int i = int.Parse( Console.ReadLine() );
 Console.Write("Kerem a matrix oszlopat [1..{0}]:",M);
 int j = int.Parse( Console.ReadLine() );
 Console.Write("Kerem az erteket:");
 int x = int.Parse( Console.ReadLine() );
 if (1<=i && i<=N && 1<=j && j<=M)
 {
   m[i-1,j-1] = x;
   db++;
 }
 else Console.Write("Na ezt megegyszer, hibas volt.");
}
```

9.28. forráskód. Adatbekérés koordinátákkal v2.0

```
for (int db=0;db<<N*M;db++)
{
 // sor
 int i;
 do
 {
   Console.Write("Kerem a matrix sorat [1..{0}]:",N);
   i = int.Parse( Console.ReadLine() );
 }
 while (i<1 || i>N);
 // oszlop
 int j;
 do
 {
   Console.Write("Kerem a matrix oszlopat [1..{0}]:",M);
```

```
   j = int.Parse( Console.ReadLine() );
 }
 while (j<1 || j>M);
 // ertek
 Console.Write("Kerem az erteket:");
 int x = int.Parse( Console.ReadLine() );
 // tarolas
 m[i-1,j-1] = x;
}
```

9.29. forráskód. Adatbekérés függvények segítségével v3.0

```
static int sorBekeres()
{
int i;
Console.Write("Kerem a matrix sorat [1..{0}]:",N);
do
{
   i = int.Parse( Console.ReadLine() );
   if (i<1 || i>N) Console.WriteLine("Nem jo. Ujra!");
}
while (i<1 || i>N);
return i;
}
         //.............................................................
static int oszlopBekeres()
{
Console.Write("Kerem a matrix oszlopat [1..{0}]:",M);
while(true)
{
   int j = int.Parse( Console.ReadLine() );
   if (1<=j && j<=M) return j;
   else Console.WriteLine("Nem jo. Ujra!");
}
}
         //.............................................................
static void Main()
{
 ...
 for (int db=0;db<<N*M;db++)
 {
   int i=sorBekeres();
   int j=oszlopBekeres();
   Console.Write("Kerem az erteket:");
   int x = int.Parse( Console.ReadLine() );
   // tarolas
   m[i-1,j-1] = x;
 }
}
```

9.30. forráskód. String felbontása a Split segítségével

```
// pl "12,24,35"
string s = Console.ReadLine();
string[] elemek = s.Split(',');
// ez esetben az elemek vektor 3 elemű lesz
// elemek[0] = "12"
// elemek[1] = "24"
// elemek[2] = "35"
```

9.31. forráskód. Console.ReadKey használata

```
ConsoleKeyInfo k = Console.ReadKey();
if (k.KeyChar == 'n' || k.KeyChar == 'N')
 Console.Write("NEM");
else if (k.KeyChar == 'i' || k.KeyChar == 'I')
 Console.Write("IGEN");
```

9.32. forráskód. Mátrix előfeltöltése speciális értékekkel

```
for (int i = 0; i < N; i++)
 for (int j = 0; j < M; j++)
   t[i] = int.MinValue;
```

9.33. forráskód. Mátrix cellája kitöltött-e

```
class Program
{
 const int N = 5;
 const int M = 4;
 public static void Main()
 {
   // matrix letrehozasa
   int[,] m = new int[N,M];
   // cellkitoltott matrix letrehozasa
   bool[,] b = new bool[N,M];
 }
}
```

9.34. forráskód. Kitöltetlen elemek száma

```
static int kitoltetlenElemekSzama()
```

```
{
  int db=0;
  for(int i=0;i<N;i++)
   for (int j=0;j<M;j++)
     if (seged[i,i]==false) db++;
  //
  return db;
}
```

9.35. forráskód. A terület elővizsgálata

```
int x   = int.Parse(Console.ReadLine());
int y   = int.Parse(Console.ReadLine());
int szel = int.Parse(Console.ReadLine());
int mag = int.Parse(Console.ReadLine());
int x   = int.Parse(Console.ReadLine());
// ---- ELOZETES KORREKCIO -----
// x negativ, kilog balra
if (x<0) { szel = szel+x; x=0;}
// x kilog jobbra
if (x>N-1) { szel=0; }
// y negativ, kilog fent
if (y<0) { mag = mag+y; x=0;}
// y kilog lent
if (y>=M) { mag=0; }
// x+szel kilog jobbra
if (x+szel>N-1) szel=N-x;
// y+mag kilog alul
if (y+mag>M-1) mag=M-y;
// ----- FELTOLTES ----
for(int i=x;i<x+szel;i++)
  for(int j=y;j+mag;j++)
   m[i,j]=x;
```

9.36. forráskód. A koordináták egyenkénti ellenőrzése

```
int x   = int.Parse(Console.ReadLine());
int y   = int.Parse(Console.ReadLine());
int szel = int.Parse(Console.ReadLine());
int mag = int.Parse(Console.ReadLine());
int x   = int.Parse(Console.ReadLine());
// ----- FELTOLTES ----
for(int i=x;i<x+szel;i++)
  for(int j=y;j+mag;j++)
   if (0<=i && i<N && 0<=j && j<M)
     m[i,j]=x;
```

9.37. forráskód. Navigálás a mátrixban

```
const int N = 5;
const int M = 4;
int[,] m = new int[N, M];

for (int k = 0; k < N; k++)
{
   for (int l = 0; l < M; l++)
      Console.Write(" {0,5} ", m[k, l]);
   Console.WriteLine();
}

int i = 0, j = 0;
bool folyt = true;
while (folyt)
{
   // m[i,j] cella kiemelese
   Console.SetCursorPosition(j * 7, i);
   Console.BackgroundColor = ConsoleColor.Green;
   Console.Write(" {0,5} ", m[i, j]);
   Console.BackgroundColor = ConsoleColor.Black;
   //
   ConsoleKeyInfo k = Console.ReadKey();
   // m[i,j] cella kiemeles megszuntetese
   Console.SetCursorPosition(j * 7,i );
   Console.BackgroundColor = ConsoleColor.Black;
   Console.Write(" {0,5} ", m[i, j]);
   // ..
   switch (k.Key)
   {
      case ConsoleKey.UpArrow:
         if (i > 0) i--;
         break;
      case ConsoleKey.DownArrow:
         if (i < N - 1) i++;
         break;
      case ConsoleKey.LeftArrow:
         if (j > 0) j--;
         break;
      case ConsoleKey.RightArrow:
         if (j < M - 1) j++;
         break;
      case ConsoleKey.Escape:
         folyt = false;
         break;
      case ConsoleKey.Enter:
         // m[i,j] bekerese
         // ...
         break;
   }
}
```

9.38. forráskód. Feltöltés véletlen számokkal

```
Random rnd = new Random();
//
for (int i = 0; i < N; i++)
{
   for (int j = 0; j < M; j++)
      m[i, j] = rnd.Next(A,B+1);
}
```

9.39. forráskód. Beolvasás fájlból

```
using System;
using System.Text;
using System.IO;

string fname = @"c:\adatok.txt";
StreamReader r = new StreamReader(fname, Encoding.Default);
while (!r.EndOfStream)
{
 string s = r.ReadLine();
 ...
}
r.Close();
```

9.40. forráskód. Beolvasás fájlból

```
StreamReader r = new StreamReader(fname, Encoding.Default);
// elso sor N es M erteke
string[] ee = r.ReadLine().Split(' ');
int N = int.Parse( ee[0] );
int M = int.Parse( ee[1] );
int[,] m = new int[N,M];
// a matrix sorainak beolvasas
int i=0;
while (!r.EndOfStream)
{
 string[] ss = r.ReadLine().Split(' ');
 for(int j=0;j<M;j++)
   m[i,j] = int.Parse(ss[j]);
 i++;
}
r.Close();
```

9.41. forráskód. Labirintus beolvasása fájlból

```
StreamReader r = new StreamReader(fname, Encoding.Default);
// elso sor N erteke
int N = int.Parse( Console.ReadLine() );
string[] m = new string[N];
for(int i=0;i<N;i++)
{
 m[i] = r.ReadLine();
}
r.Close();
```

9.42. forráskód. Labirintus kiírása a képernyőre

```
for(int i=0;i<N;i++)
{
 for(int j=0;j<m[i].Length;j++)
 {
   if (m[i][j]=='.') Console.ForegroundColor = ConsoleColor.Green;
   else Console.ForegroundColor = ConsoleColor.Red;
   Console.Write(m[i][j]);
 }
 Console.WriteLine();
}
```

9.43. forráskód. A vastagítás kódja (1. rész)

```
const int N = 20;
static int[,] m = new int[N, N];
static Random rnd = new Random();
//
static void vulkanKratere(int a, int f)
{
   int x = (N / 2) + rnd.Next(-1,+2);
   int y = (N / 2) + rnd.Next(-1, +2);
   m[x, y] = rnd.Next(a, f + 1);
}
```

9.44. forráskód. A vastagítás kódja (2. rész)

```
static bool szomszedSzimpatikus(int x, int y)
{
   for (int i = x - 1; i <= x + 1; i++)
      for (int j = y - 1; j <= y + 1; j++)
         if (0 <= i && i < N && 0 <= j && j < N && m[i, j] > 0)
            return true;
   //
```

```
   return false;
}
        //.....................................................
static void vastagit(int a, int f, int esely)
{
   int[,] temp = (int[,])(m.Clone());
   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
         if (temp[i,j]==0 && szomszedSzimpatikus(i, j))
            if (rnd.Next(0,100)<esely)
               temp[i, j] = rnd.Next(a, f + 1);
   m = temp;
}
```

9.45. forráskód. A vastagítás kódja (3. rész)

```
static void Main()
{
   vulkanKratere(200, 300);
   for(int i = 0;i<2;i++)
      vastagit(400, 500, 70-i*10);
   for (int i = 0; i < 2; i++)
      vastagit(300, 400, 70 - i * 10);
   for (int i = 0; i < 2; i++)
      vastagit(200, 300, 70 - i * 10);
   for (int i = 0; i < 2; i++)
      vastagit(100, 200, 70 - i * 10);

   //
   for (int i = 0; i < N; i++)
   {
      for (int j = 0; j < N; j++)
      {
         ConsoleColor c = ConsoleColor.Gray;
         if (m[i, j] < 100) c = ConsoleColor.Gray;
         else if (m[i, j] < 200) c = ConsoleColor.Yellow;
         else if (m[i, j] < 300) c = ConsoleColor.Green;
         else if (m[i, j] < 400) c = ConsoleColor.Blue;
         else if (m[i, j] < 500) c = ConsoleColor.Red;
         Console.ForegroundColor = c;
         if (m[i, j] == 0) Console.Write('.');
         else Console.Write('#');
      }
      Console.WriteLine();
   }
   Console.ReadKey();
}
```

9.46. forráskód. Festő algoritmus

```
static void befest(int x, int y)
{
   if (m[x, y] != 0) return;
   m[x, y] = 1;
   if (x > 0) befest(x - 1, y);
   if (x <N-1) befest(x + 1, y);
   if (y > 0) befest(x, y-1);
   if (y < N - 1) befest(x, y+1);
}
        //.....................................................
static void jolBefest()
{
   for (int i = 0; i < N; i++)
   {
      if (m[i, 0] == 0) befest(i, 0);
      if (m[0, i] == 0) befest(0, i);
      if (m[i, N-1] == 0) befest(i,N-1);
      if (m[N - 1,i] == 0) befest(N - 1,i);
   }
}
```

9.47. forráskód. Korrekciózás a belső cellákra

```
static int atlagSzamit(int x, int y)
{
   int ossz = 0, db = 0;
   for (int i = x - 1; i <= x + 1; i++)
      for (int j = y - 1; j <= y + 1; j++)
         if (0 <= i && i < N && 0 <= j && j < N && m[i, j] > 0)
         {
            ossz += m[i];
            db++;
         }
   if (db == 0) return 0;
   else return ossz/db;
}
        //.....................................................
static void korrekcio()
{
   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
         if (m[i, j] == 0)
         {
            int x = atlagSzamit(i, j)+rnd.Next(-20,20);
            if (x<0) x=0;
            else if (x>500) x=500;
            m[i, j] = x;
         }
}
```

10. fejezet - Numerikus műveletek mátrixokkal (szerző: Hernyák Zoltán)

Tartalom

Az előző fejezetben több mint 20 módszert vettünk mátrixok feltöltésére. Ezen fejezetben a kiindulási alap, hogy egy vagy két, akár különböző méretű mátrixunk valamilyen módon fel van töltve elemekkel. Javasolt fájlból feltölteni, mert úgy könnyen módosíthatóak a mátrix értékei, és könnyű újra futtatni a programot egy hibás működés felfedezése és a javítás után. De a véletlen értékekkel történő feltöltés is megfelelő lehet. A mátrixot feltöltése után mindig írassuk ki a képernyőre táblázatos alakban, hogy lássuk a feltöltés működését!

A feltöltött mátrixokkal kapcsolatosan sok érdekes és kevésbé érdekes matematikai, numerikus jellegű probléma oldható meg. Ezen alapvető mátrixműveletekre sok programozási probléma vezethető vissza, ezért érdemes őket elkészíteni. A megoldás során a ciklusok gyakorlására is lehetőségünk van.

Bevezető információk: Amennyiben egy *A* ($N \times N$ méretű) mátrix elemeit tükrözzük a főátlóra, úgy a kapott mátrixot az eredeti *A* mátrix *transzponáltjának* nevezzük.

10.1. feladat (Tükrözés a főátlóra – *szint: 2*). Egy $N \times N$ méretű négyzetes mátrixot töltsünk fel véletlen értékekkel, majd jelenítsük meg a képernyőn táblázatos alakban! Készítsük el ezen mátrix transzponáltját! Készítsük el ezt a programot úgy, hogy két mátrixváltozóval dolgozunk, az eredeti (*A*) mátrix elemeit nem módosítjuk, a tükrözött értékeket egy második, egyező méretű mátrixba (*B*) generáljuk le! Jelenítsük meg ezen *B* mátrixot is a képernyőn táblázatos alakban!

Magyarázat: A megoldáshoz egy egymásba ágyazott (*dupla*) for ciklus elég. Arra kell csak ügyelni, hogy ne menjen mindkét ciklus $[0...N-1]$-ig, mert ha az a[i,j] cellát felcseréljük az a[j,i] cellával, miközben az i = 2, j = 3, akkor ne kerüljön ugyanezen cserére sor i = 3, j = 2 értékek esetén is, mert akkor az előző cserét „visszacseréljük", végeredményképp a mátrix celláiban az eredeti értékek maradnak, a mátrix változatlan lesz.

10.2. feladat (Tükrözés a mellékátlóra – *szint: 3*). Egy $N \times N$ méretű négyzetes mátrixot töltsünk fel véletlen értékekkel, majd jelenítsük meg a képernyőn táblázatos alakban! Helyben tükrözzük ezt a mátrixot a mellékátlójára, vagyis ne készítsünk el egy második, egyező méretű mátrixot! A tükrözés során az eredeti mátrixban hajtsuk végre a módosításokat! Jelenítsük meg az eredményül kapott értékeket is táblázatos alakban!

Magyarázat: Ez az előző feladathoz nagyon hasonló a feladat, de vagy a ciklusok futási intervallumait kell ügyesebben beállítani, vagy egyfajta transzformációt kell végezni a koordinátákon.

Bevezető információk: Egységmátrixnak nevezzük azt az $N \times N$ méretű négyzetes mátrixot, amelynek minden eleme 0, kivéve a főátló elemeit, amelyeknek értéke 1.

10.3. feladat (Egységmátrix készítése – *szint: 1*). Készítsünk el egy $N \times N$ méretű egységmátrixot, ahol *N* méretét a felhasználó adja meg! A feltöltött mátrixot jelenítsük meg a képernyőn!

Magyarázat: Minden a[i,j] cellába 0 értéket kell helyezni, kivéve ha *i==j* teljesül (átlóba eső cella), ahova 1-et kell berakni.

10.4. feladat (Egységmátrix-e – *szint: 1*). Egy $N \times N$ méretű mátrixról döntsük el, hogy egységmátrix-e.

Magyarázat: Megoldható a feladat úgy is, hogy generálunk egy $N \times N$ méretű egységmátrixot a 10.3 feladatban leírtak szerint, majd meg tudjuk vizsgálni celláról cellára, hogy minden elem egyenlő-e a generált mátrix megfelelő cellájával. Mivel azonban az egységmátrix előállítása nagyon egyszerű szabályok mentén történik, így a vizsgálat során futás közben is generálható az aktuálisan ellenőrzött cella kívánt értéke.

Bevezető információk: Két (*A* és *B*) egyforma méretű ($N \times M$) mátrixok összegén értsünk egy harmadik *C* mátrixot, mely szintén $N \times M$ méretű, s melynek elemeit úgy kell kiszámítani, hogy az *A* és *B* mátrixok egyező koordinátájú celláiban lévő értékeket kell összeadni!

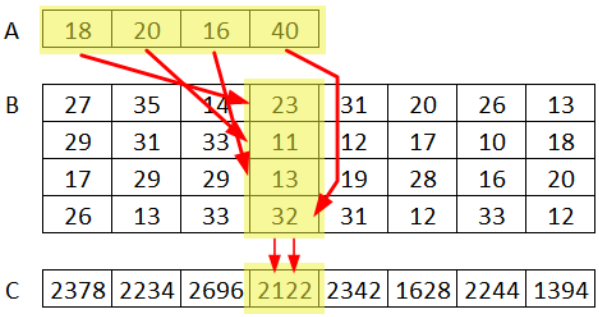
10.5. feladat (Mátrixok összeadása – *szint: 2*). Legyen két $N \times M$ méretű mátrixunk! Készítsük el a két mátrix *összegét*! A program jelenítse meg a mátrixokat táblázatos formában – egyszerre csak egy mátrixot –, melyek között a *Page Up* és *Page Down* billentyűkkel lehessen lapozni! A program az *Esc* leütésére lépjen ki!

Magyarázat: A *C* mátrix esetén a C[i,j] = A[i,j] + B[i,j] képlet segítségével kell módszeresen minden cellát feltölteni. Egy egymásba ágyazott cikluspár elegendő a feladathoz.

10.6. feladat (Mátrix szorzása vektorral – *szint: 3*). Legyen egy *N* méretű vektorunk és egy $M \times N$ méretű mátrixunk! Készítsük el a vektor és a mátrix szorzatát (*M* méretű vektor lesz)! A program jelenítse meg a vektort, a mátrixot táblázatos formában, majd az eredményt is! A megjelenítési lehetőségek között a *Page Up* és *Page Down* billentyűkkel lehessen lapozni! A program az *Esc* leütésére lépjen ki!

Magyarázat: A vektor és mátrix szorzása két egymásba ágyazott ciklussal valósítható meg. A vektor minden elemét meg kell szorozni a mátrix megfelelő oszlopában lévő elemekkel (10.4. forráskód) (pl. 2122 = 18 * 23 + 20 * 11 + 16 * 13 + 40 * 32).

10.1. ábra. A C = A * B vektor szorzása mátrixszal – magyarázó ábra



10.1. forráskód. A C=A*B vektor szorzása mátrixal

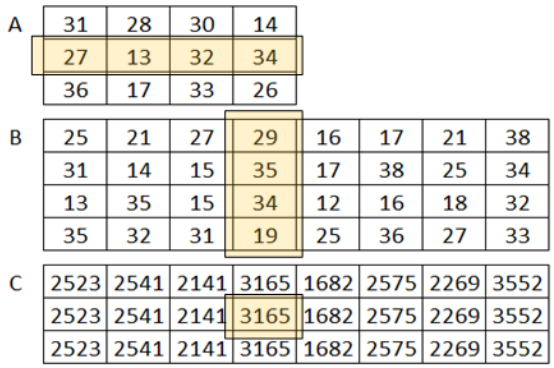
```
for(int i=0;i<M;i++)
{
  // A vektor * B matrix i. soraval
  int sz = 1;
  for(int j=0;j<N;j++)
```

```
    sz = sz*A[i]*B[i,j];
   // tarolas
   C[i]=sz;
}
```

10.7. feladat (Mátrix szorzása mátrixszal – *szint: 4*). Legyen egy $N \times M$ és egy $M \times K$ méretű mátrixunk! Készítsük el a két mátrix szorzatát ($N \times K$ méretű lesz)! A program jelenítse meg a mátrixokat táblázatos formában, egyszerre csak egy mátrixot, melyek között a *Page Up* és *Page Down* billentyűkkel lehessen lapozni! A program az *Esc* leütésére lépjen ki!

Magyarázat: A mátrixok szorzása három egymásba ágyazott ciklussal valósítható meg. A célmátrix [i,j] koordinátájú elemét úgy kell kiszámolni, hogy az A mátrix i. sorát kell összeszorozni a B mátrix j. oszlopában szereplő elemekkel.

10.2. ábra. A C = A * B vektor szorzása mátrixszal – magyarázó ábra



Bevezető információk: Egy $N \times N$ méretű *A* mátrix inverzének nevezzük azt az egyező méretű *K* mátrixot, melyre teljesül az $A \cdot K = E$, és ugyanekkor az $K \cdot A = E$ is, ahol E az egységmátrix. Az *A* mátrix inverz mátrixának jele A^{-1}.

10.8. feladat (Inverzmátrix-ellenőrzés – *szint: 3*). Olvassunk be egy text fájlból két egyforma méretű *A* és *B* kvadratikus mátrixot! Ellenőrizzük le, hogy a *B* mátrix az *A* mátrix inverze-e!

Magyarázat: A 10.7 feladatban leírtak szerint kiszámíthatóak a szorzatmátrixok ($A \cdot K$ és $K \cdot A$), és a 10.4 feladatban megadottak szerint eldönthető, hogy egységmátrix-e.

Bevezető információk: Egy *A* mátrix *nyeregpontjának* nevezzük azt az elemét, amely a legkisebb a sorában és legnagyobb az oszlopában. Egy mátrixban több nyeregpont is lehet, de elképzelhető, hogy egy sem található.

10.9. feladat (Nyeregpontok – *szint: 2*). Készítsünk olyan programot, amely egy $N \times M$ méretű mátrix adatait beolvassa fájlból, majd megjeleníti azokat a képernyőn táblázatos formában! A mátrix nyeregpontjait a program színezze be sárgára a táblázatban! A mátrix alatt soroljuk fel a nyeregpontok koordinátáit!

Magyarázat: Amennyiben készítünk egy-egy függvényt, amely képes valamely i. sorban meghatározni a legkisebb elem értékét, valamint egy j. oszlop legnagyobb elemének értékét, úgy (erőforrás-pazarló módon bár) könnyű a nyeregpontokat megtalálni (lásd a 10.5. forráskód).

10.2. forráskód. A nyeregpontok keresése - v1.0 vázlatos

```
for(int i=0;i<N;i++)
{
 for(int j=0;j<N;j++)
 {
    if (sorMinimum(i)==oszlopMaximum(j))
      nyeregPont(i,j);
 }
}
```

Másrészt a 10.5. kódban ugyanazon i. sorbeli minimumot *M*-szer számoljuk ki, hasonlóan, egy j. oszlopbeli maximumot *N*-szer számolunk ki. Célszerűbb a minimumokat egyszer kikalkulálni, és egy *N* elemű, a maximumokat pedig egy *M* méretű vektorban eltárolni. Később könnyű ezek egyenlőségét még megvizsgálni (10.6. forráskód).

10.3. forráskód. A nyeregpontok keresése - v2.0 vázlatos

```
for(int i=0;i<N;i++)
 minimum[i] = sorMinimum(i);
for(int j=0;j<N;j++)
 maximum[j] = oszlopMaximum(j)
// ------------------------------------
for(int i=0;i<N;i++)
{
 for(int j=0;j<N;j++)
 {
    if (minimum[i]==maximum[j])
      nyeregPont(i,j);
 }
}
```

Bevezető információk: Egy *A* mátrix *paritáspontjának* nevezzük azt az elemét, amelyre igaz, hogy a sorában lévő elemek összege páros, az oszlopában lévő elemeké pedig páratlan. Hasonlóan a nyeregponthoz, egy mátrixban több paritáspont is létezhet, de előfordulhat, hogy egy sincs benne.

10.10. feladat (Paritáspontok – *szint: 2*). Készítsünk olyan programot, amely egy $N \times M$ méretű mátrix adatait beolvassa fájlból, majd megjeleníti azokat a képernyőn táblázatos formában! A mátrix paritáspontjait a program színezze be sárgára a táblázatban! Adjuk meg a paritáspontok számát!

Magyarázat: A 10.9-es feladatban leírtaknak megfelelő módon a probléma kezelhető, csak a sorminimum- és oszlopmaximum-függvények helyett a sor- és

oszlopösszeg-számító függvényeket kell használni. A számított összegek paritását a kettővel való osztási maradékokból lehet megállapítani. Ha *s* egy ilyen összeg, akkor az *s%2* művelet adja meg a kettővel való osztási maradékot.

A fejezet forráskódjai

10.4. forráskód. A C = A * B vektor szorzása mátrixszal

```
for(int i=0;i<M;i++)
{
  // A vektor * B matrix i. soraval
  int sz = 1;
  for(int j=0;j<N;j++)
   sz = sz*A[i]*B[i,j];
  // tarolas
  C[i]=sz;
}
```

10.5. forráskód. A nyeregpontok keresése – v1.0 vázlatos

```
for(int i=0;i<N;i++)
{
 for(int j=0;j<N;j++)
 {
    if (sorMinimum(i)==oszlopMaximum(j))
      nyeregPont(i,j);
 }
}
```

10.6. forráskód. A nyeregpontok keresése – v2.0 vázlatos

```
for(int i=0;i<N;i++)
 minimum[i] = sorMinimum(i);
for(int j=0;j<N;j++)
 maximum[j] = oszlopMaximum(j)
// -----------------------------------
for(int i=0;i<N;i++)
{
 for(int j=0;j<N;j++)
 {
    if (minimum[i]==maximum[j])
      nyeregPont(i,j);
 }
}
```

11. fejezet - Mátrixok vizsgálata (szerző: Hernyák zoltán)

Tartalom

11.1. feladat (Amőbanyertes – *szint: 4*). Tételezzük fel, hogy egy két játékos *amőba* játékot játszik egy $N \times N$-es területen! A játék egy állapotát gépre viszik. Ezen mátrixban csak 0, 1, 2 értékek fordulnak elő. A 0 érték jelöli, hogy a cella üres, az 1 jelöli, hogy a cellában x, a 2 pedig hogy a cellában o szimbólum van. A program feladata eldönteni, hogy mi a játék aktuális állása szerint a helyzet. Elképzelhető, hogy az egyik játékos nyert? Vagy egyik sem nyert? Extrém (rosszul) kitöltött mátrix esetén több nyerő helyzet (5 x vagy 5 o) is szerepelhet a mátrixban, vagy van esetleg 5-nél több is vonalban vagy átlóban.

A program a mátrix értékeit egy text fájlból olvassa fel, melyben soronként az amőba játék táblázatának egy-egy sora szerepel. A sorokban ténylegesen x és o karakterek szerepelnek, az üres cella helyén pedig a . karakter. Ez alapján építsük fel a mátrixot, végezzük el az ellenőrzést, majd jelenítsük meg a mátrixot oly módon a képernyőn, hogy a nyerést jelentő 5 db x pirossal, az 5 db o karakter pedig zöld színnel jelenjen meg, a többi része a mátrixnak legyen szürke színű!

A program ellenőrizze és jelezze ki a győztest! Három lehetőség van:

- a játékállás szerint nincs nyertes,
- a játékállás szerint pontosan egy 5-ös csoport van, amely az amőba játék szabályai szerint egy vonalban (vagy egy átlóban) szerepel egymás mellett – az egyik játékos megnyerte,
- a mátrix extrém, rosszul van kitöltve.

Magyarázat: Számoljuk meg, hány pontosan 5 egység hosszú sor, oszlop vagy átlósan elhelyezkedő elemet találunk a mátrixban! A megszámolást kezdeményezzük minden lehetséges pozícióból kiindulva, de elég csak jobbra, le és jobbra-le átlósan végezni. Ha találunk 5 szomszédos egyforma jelet, akkor abba is hagyhatjuk a számolást. Nyilvánvaló, hogy ha van a mátrixban 5-ös nyerő sorozat, akkor így meg fogjuk találni, és pontosan egyszer fogunk (erre) rátalálni. Ha lesz benne 6-os (vagy hosszabb) sorozat, akkor ezen módszerrel több 5-ös sorozatot is fogunk találni, ami elégséges ahhoz, hogy megállapítsuk a mátrix extrém kitöltésének tényét (11.16. forráskód).

11.1. forráskód. Amőba nyertesek megszámolása

```
odb = 0;
xdb = 0;
for(int i=0;i<N;i++)
{
for(int j=0;j<N;j++)
{
  // x-t tartalmazo cella
  if (m[i,j]==1)
  {
   if (sorban5(i,j))   xdb ++;
```

```
   if (oszlopban5(i,j)) xdb ++;
   if (atlosan5(i,j))  xdb ++;
  // o-t tartalmazo cella
  if (m[i,j]==2)
  {
   if (sorban5(i,j))   odb ++;
   if (oszlopban5(i,j)) odb ++;
   if (atlosan5(i,j))  odb ++;
  }
}
}
```

Amennyiben az *odb* vagy *xdb* változók egyike 1, a másika 0 értéket tartalmaz, az egyik megnyerte. Ha az *odb* + *xdb* értéke 0, akkor egyiknek sincs 5-ös nyerő sora, senki sem nyert. Ha az *odb* + *xdb* > *1*, akkor valami gond van a mátrixban, így extrém kitöltés állapítható meg.

11.2. feladat (Alul vagy felül nagyobb – *szint: 3*). Egy $N \times N$ méretű mátrix elemeit olvassuk be fájlból! Keressük meg az alsó és a felső háromszögmátrix legnagyobb elemét! Ezen elemeket jelöljük meg más színnel a képernyőn! Adjuk meg, alul vagy felül van-e a legnagyobb elem, de vegyük figyelembe, hogy a két érték lehet egyenlő is! A háromszögekbe ez esetben a főátlóbeli elemek nem kerülnek be.

Magyarázat: Az alsó háromszögmátrix a második sorban kezdődik, és 1 oszlop széles. A maximum keresés során célszerű pozíciót keresni, mivel a táblázatos megjelenítés folyamán majd ezt az értéket ki kell emelni más színnel (11.1. forráskód). A felső háromszögmátrix ehhez hasonlóan kezelhető.

11.1. ábra. A program outputjának terve

	1	2	3	4	5	6	7	8	9	10
1	17	59	93	72	63	62	69	31	67	23
2	22	35	14	24	38	26	28	97	64	95
3	19	86	30	36	30	52	93	43	97	70
4	44	68	16	93	98	24	75	70	57	54
5	29	99	25	48	81	17	90	77	29	30
6	63	99	59	45	46	34	96	23	28	21
7	25	96	39	97	45	20	39	58	43	90
8	17	57	11	44	97	97	24	88	89	62
9	30	77	58	16	45	71	63	96	61	95
10	57	64	87	16	29	59	94	56	87	14

Az alsó háromszögmátrixban van a nagyobb szám.

11.2. forráskód. A maximumkeresés vázlata

```
ahmx=2;
ahmy=1;
for(int i=1;i<N;i++)
for(int j=0;j<i;j++)
{
  if (m[ahmy,ahmx]<m[i,j])
  {
     ahmx = i;
     ahmy = j;
  }
}
```

Bevezető információk:

- Egy mátrixot *diagonális* mátrixnak nevezünk, ha csak a főátlójában van nullától különböző (de nem feltétlenül 1 értékű) elem. Nem szükséges, hogy a főátlóban minden elem különbözzön a nullától, sőt, az sem, hogy a főátlóban legyen egyáltalán nullától különböző elem. Az szükséges viszont, hogy azok az elemek, amelyek nem a főátlóban vannak, garantáltan zéró értékűek legyenek.
- Azt a speciális diagonális mátrixot, ahol a főátlóban csupa 1 érték szerepel, *egységmátrixnak* nevezzük.
- Egy mátrixot *nullmátrixnak* nevezünk, ha minden eleme 0.

11.3. feladat (Egységmárix, nullmátrix – *szint: 2*). Írjunk olyan programot, amely beolvas egy mátrixot fájlból, majd megvizsgálja, hogy nullmátrix-e, egységmátrix-e, diagonális mátrix-e, vagy egyéb (közönséges) mátrix! A megfelelő szöveget írjuk ki a képernyőre (egyszerre csak egy kiírás történhet meg, a mátrix legjellemzőbb tulajdonságát írjuk ki)!

Magyarázat: A feladat megoldása inkább munkás, mint nehéz. A nullmátrix vizsgálata egyszerű: minden cellában 0 értéknek kell szerepelnie. Az egységmátrix vizsgálatának problémáját már a 3.4 feladat tartalmazta. Hogy diagonális mátrix-e egyáltalán, ahhoz ellenőrizzük, hogy a főátlón kívül van-e olyan cella, amelyben nem nulla van – érdemes külön megszámolni, hány nem nulla értékű elem van a főátlóban, és hány nem nulla van a főátlón kívül (11.18. forráskód).

11.3. forráskód. A nem nulla elemek megszámolása

```
foatlo_db=0;
egyeb_db=0;
for(int i=0;i<N;i++)
{
 for(int j=0;j<N;j++)
 {
   if (m[i,j]>0 && i==j) foatlo_db++;
   else if (m[i,j]>0 && i!=j) egyeb_db++;
 }
}
```

Bevezető információk: Egy mátrix

- *szimmetrikus* (felső háromszög), ha a főátlóra nézve szimmetrikus elemek egyenlőek,

- *ferdeszimmetrikus*, ha a főátlóra nézve szimmetrikus elemek egyenlőek, de ellenkező előjelűek.

11.4. feladat (Szimmetrikusság – *szint: 2*). Írjunk olyan programot, amely beolvas egy mátrixot fájlból, majd megvizsgálja, hogy szimmetrikus, ferdeszimmetrikus vagy egyéb (közönséges) mátrix-e! A megfelelő szöveget írjuk ki a képernyőre!

Magyarázat: A szimmetrikus elemek indexe m[i,j] vs. m[j,i]. Igazából csak hatékonysági kérdés, hogy a beágyazott ciklusok csak a háromszögmátrixon menjenek végig, és a főátlót ki is lehet hagyni akár. Ugyanakkor mivel szimmetriát kell ellenőrizni, az ellenőrzés eredményét valójában nem rontja el az sem, ha a teljes mátrixon szkenneli az i,j ciklusokat (11.19. forráskód).

11.4. forráskód. A vizsgálat nem optimalizált változata

```
szimmetrikus = true;
ferdeszimmetrikus = true;

for(int i=0;i<N;i++)
{
 for(int j=0;j<N;j++)
 {
   if (m[i,j]!=m[j,i]) szimmetrikus=false;
   if (m[i,j]!=-m[j,i]) ferdeszimmetrikus=false;
 }
}
```

Bevezető információk: Egy *A* mátrixnak a *B* mátrix inverze, ha szorzatuk maga az egységmátrix.

11.5. feladat (Inverz mátrix – *szint: 3*). Egy text fájlban két mátrix foglal helyet egymás alatt. A mátrixot leíró sorok közötti üres sor határolja a két mátrixot el egymástól. Olvassuk be a két egyforma ($N \times N$ méretű) mátrixot, és állapítsuk meg, hogy a második mátrix az első mátrix inverz mátrixa-e!

Magyarázat: A mátrixok szorzását a 3.7 feladat tárgyalja. A kapott mátrixot meg kell vizsgálni, hogy egységmátrix-e. Ezzel több feladat is foglalkozott, legkorábban a 3.4 feladat. A korábbi feladatok megoldása után ezen feladat megoldása már csak ujjgyakorlat.

Bevezető információk: Egy $N \times N$ mátrix ortogonális, ha a transzponáltja egyenlő az inverzével. Megj.: az ortogonális mátrixok írják le az egybevágósági transzformációkat az *N* dimenziós térben.

11.6. feladat (Ortogonális – *szint: 3*). Egy $N \times N$ méretű mátrixot olvassunk be fájlból, majd határozzuk meg, hogy ortogonális-e!

Bevezető információk:

- Két vektor (v_1, v_2) *lineárisan független*, ha a $\lambda_1 v_1 + \lambda_2 v_2$ lineáris kombinációjuk csak úgy lehet nullvektor, ha λ_1 és λ_2 is nulla értékű.
- Az *A* ($N \times K$ méretű) *mátrix rangja* a mátrix lineárisan független oszlopainak maximális száma. Igazolható, hogy ez egy jól definiált természetes szám, és megegyezik a mátrix lineárisan független sorainak maximális számával (a sorrang tehát egyenlő az oszlopranggal).

11.7. feladat (Mátrix rangja – *szint: 4*). Egy $N \times K$ méretű mátrixot olvassunk be fájlból, majd határozzuk meg a mátrix rangját!

Magyarázat: A vektorok lineáris függetlenségét úgy tudjuk igazolni, hogy megpróbálunk keresni megfelelő λ_1 és λ_2 párokat. Könnyű igazolni, hogy ha ilyenek léteznek, akkor $\lambda_1 = 1$ esetén is létezik megfelelő λ_2. Vagyis válasszuk $\lambda_1 = 1$ esetet, és akkor már csak a λ_2-vel kell foglalkozni. Szintén könnyű belátni, hogy ez esetben $\lambda_2 = v_1[i]/v_2[i]$ kell legyen, ahol *i* az első olyan elem sorszáma, ahol $v_2[i] \neq 0$ teljesül. Ha nincs ilyen elem, akkor a két vektor biztosan lineárisan függő (ekkor például a $\lambda_1 = 0, \lambda_2 = 1$ választással adódik a nullvektor képzése).

Tehát ha ellenőrizni szeretnénk, hogy az *m* mátrix *i*. és *j*. sora lineárisan független-e, akkor annyi teendőnk van, hogy

- megkeressük a *j*. sor első nem nulla értékű elemének sorszámát (jelöljök *k*-val),
- ha nem találunk ilyet, akkor máris megállapítjuk, hogy ezen két mátrixsor nem lineárisan független,
- ha találunk, akkor *C := m[i,k]/m[j/k]* értéket kiszámoljuk,
- ellenőrizzük, hogy minden $j \in [0...K-1]$ oszlopindexre $[m[i,j] + C * m[j,k] = 0$ teljesül-e. Ha igen, akkor a mátrix két sora lineárisan függő. Ha bármely *j* index esetén a kifejezés nem 0, akkor a két sor lineárisan független.

A mátrix rangját úgy kapjuk meg, hogy megszámoljuk, hogy a 0. sora hány más sortól lineárisan független (jelöljük r_0-val), majd megszámoljuk, hogy az 1. sora hány más sorral lineárisan független (r_1), stb. A mátrix rangja az előbbi $r_0, r_1, ..., r_{n-1}$ értékek maximuma lesz.

A 11.20. ... 11.24 forráskódok lefedik a problémát.

11.5. forráskód. A mátrix rangjának kalkulálása

```
int rang = 0;
for (int i = 0; i < N; i++)
{
   bool fuggetlen = true;
   int r_i = 0;
   for (j = 0; j < N; j++)
      if (i != j && lin_fuggetlen(i, j)) r_i++;
   if (r_i > rang) rang = r_i;
}
```

11.6. forráskód. A lineárisan függetlenség vizsgálata

```
static bool lin_fuggetlen(int i, int j)
{
 if (nullvektor(i)) return false;
 if (nullvektor(j)) return false;
 int k = elso_nem_nulla(j);
 double c = (double)m[i,k]/m[j,k];
 if (mind_nulla(i,j,c)) return true;
```

```
 else return false;
}
```

11.7. forráskód. A mátrix *i*. sora nullvektor-e

```
static bool nullvektor(int i)
{
 for(int j=0;j<K;j++)
 {
  if (m[i,j]!=0) return false;
 }
 return true;
}
```

11.8. forráskód. A mátrix *i*. sorának első nem 0 elemének oszlopindexe

```
static bool elso_nem_nulla(int i)
{
 for(int j=0;j<K;j++)
 {
   if (m[i,j]!=0) return j;
 }
 return 0;
}
```

11.9. forráskód. A mátrix *i*. és *j*. sorának *C* szerinti lineáris kombinációja nullvektor-e

```
static bool mind_nulla(int i, int j, double C)
{
 for(int j=0;j<K;j++)
 {
   if (m[i,k]+C*m[j,k]!=0) return false;
 }
 return true;
}
```

11.8. feladat (Sorok minimuma, maximuma – *szint: 3*). Egy text fájlban szereplő $N \times M$ mátrix elemeit olvassuk be, majd jelenítsük meg a képernyőn táblázatos formában! Minden sor végére írjuk ki az adott sor minimumát és maximumát! A mátrix elemeit szürke (normál) színnel írjuk, a minimumot zölddel, a maximumot piros színnel írjuk ki a sorok végére (ugyanezt megismételhetjük az oszlopokra is)!

Magyarázat: A program egy lehetséges outputjának vázlatos terve a 11.2. ábrán látható. A megoldás során célszerű kigyűjteni a sorokban szereplő minimális elemek, egy másik vektorban pedig a maximális elemek oszlopindexeit. A megfelelően színezett kiírás innentől kezdve egyszerű.

11.2. ábra. A program outputjának terve

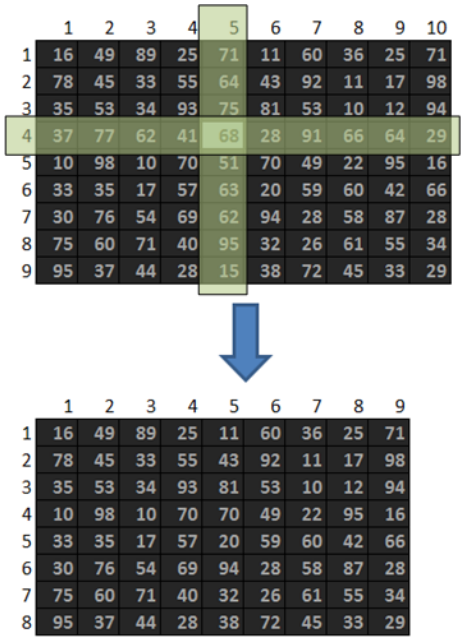
	1	2	3	4	5	6	7	8	9	10	min	max
1	99	34	80	86	83	42	70	60	29	45	29	99
2	18	36	56	71	17	31	43	96	36	65	17	96
3	41	26	86	53	30	97	10	27	15	90	10	97
4	59	33	83	83	91	84	53	62	22	48	22	91
5	56	41	35	44	13	86	30	25	46	53	13	86
6	62	94	70	47	26	12	74	56	31	16	12	94
7	74	63	84	18	68	92	51	79	31	99	18	99
8	64	80	47	90	58	74	80	79	92	75	47	92
9	64	80	85	48	49	26	55	17	68	44	17	85
10	45	54	12	37	91	22	31	69	77	71	12	91

Bevezető információk: Egy $N \times M$ méretű A mátrix i,j koordinátájú eleméhez tartozó *adjungált* mátrixnak tekintjük azt az $N-1 \times M-1$ méretű A_{ij} mátrixot, melynek elemeit úgy kapjuk, hogy az eredeti A mátrix elemeiből elhagyjuk az i. sorban és j. oszlopban szereplő elemeket.

11.9. feladat (Adjungált mátrix előállítása – *szint: 3*). Egy fájlból olvassuk be egy A ($N \times M$ méretű) mátrix értékeit, majd jelenítsük meg a képernyőn táblázatos formában! Kérjük be billentyűzetről egy sor és egy oszlop értéket ($i \in [0 \dots N-1], j \in [0 \dots M-1]$), majd generáljuk le az A_{ij} adjungált mátrixot, és jelenítsük meg a képernyőn!

Magyarázat: A mátrix elemenkénti másolásával a probléma kezelhető (a 11.25. forráskód). Egy *m* mátrix esetén C# nyelven az *m.GetLength(0)* módon tudjuk az egyik dimenzióbeli méretet, *m.GetLength(1)* módon a másik dimenzióbeli méretet lekérdezni (11.25. forráskód).

11.3. ábra. Az adjungált mátrix képzése



	1	2	3	4	5	6	7	8	9
1	16	49	89	25	11	60	36	25	71
2	78	45	33	55	43	92	11	17	98
3	35	53	34	93	81	53	10	12	94
4	10	98	10	70	70	49	22	95	16
5	33	35	17	57	20	59	60	42	66
6	30	76	54	69	94	28	58	87	28
7	75	60	71	40	32	26	61	55	34
8	95	37	44	28	38	72	45	33	29

11.10. forráskód. Az adjungált mátrix előállítása

```
static int[,] adjungalt(int[,] m, int i, int j)
{
   int N = m.GetLength(0);
   int M = m.GetLength(1);
   int[,] ret = new int[N - 1, M - 1];
   int p = 0;
   for (int k = 0; k < N; k++)
   {
      if (k != i)
      {
         int q = 0;
         for (int l = 0; l < M; l++)
         {
            if (l != j)
            {
               ret[p, q] = m[k, l];
               q++;
            }

         }
         p++;
      }

   }
   return ret;
}
```

Bevezető információk: Egy 1×1 méretű mátrix determinánsa maga az egyetlen elem értékével egyenlő. A nagyobb méretű mátrixok determinánsát valamely sorának kifejtésével nyerhetjük ki, miközben 1-el kisebb méretű (adjungált) mátrixok determinánsával dolgozunk. A 2×2 méretű mátrixok determinánsszámítását tehát az 1×1 méretűek determinánsára vezetjük vissza. Hasonlóan, a 3×3 méretű mátrixok esetén 2×2-es mátrixok determinánsát kell kiszámítani. A kifejtési tétel tehát egy *rekurzív* definíció.

A kifejtési tétel szerint válasszuk ki a mátrix tetszőleges sorát, majd ezen sorban szereplő értékeket szorozzuk fel az adott értékhez tartozó adjungált mátrix determinánsával! Ha nem az első sor szerint fejtjük ki, akkor a szorzatbeli tagok előjelére is figyelni kell. Ezért javasoljuk az első sor szerinti kifejtést, melynél az előjelek az alábbiak szerint alakulnak:

$$detA = (-1)^0 * a_{i1}A_{i1} + (-1)^1 * a_{i2}A_{i2} + \cdots + (-1)^{n-1} * a_{in}A_{in}$$

11.10. feladat (Determináns kifejtési tétellel – *szint: 4*). Olvassunk be fájlból egy $N \times M$ méretű mátrixot, majd számoljuk ki a mátrix determinánsát a kifejtési tétel értelmében!

A mátrix bármely sorát kiválaszthatjuk, amely alapján kiszámolhatjuk a kifejtésti tétel értelmében a determinánsát. Különösebb ok hiányában az első sorának választása megfelelő lehet. Ugyanakkor a kisebb méretű mátrix még mindig túl nagy méretű lehet, hogy közvetlenül kiszámítsuk a determinánsát. A 2×2 és 1×1 méretű mátrixok kivételével tovább kell alkalmazni a kifejtési tételt a determináns kiszámíthatósága miatt. Ezért rekuziót kell alkalmazni (11.26. forráskód).

11.11. forráskód. A determináns kiszámítása

```
static double determinans(int[,] m)
{
   int N = m.GetLength(0);
   if (N == 1) return m[0, 0];
   double ret = 0.0;
   int elojel = 1;
   for (int i = 0; i < N; i++)
   {
      int[,] adj = adjungalt(m, 0, i);
      double d = determinans(adj);
      ret = ret + elojel*m[0, i] * d;
      elojel = -elojel;
   }
   return ret;
```

```
}
```

Bevezető információk: Speciális felépítésű mátrixok esetén a determináns számítása egyszerűsíthető. Amennyiben pl. a mátrixunk alsó (vagy felső) háromszögmátrixában mindenütt a 0 érték szerepel, úgy a determináns értéke a főátlóbeli elemek szorzataként is kiszámítható.

11.11. feladat (Háromszögmátrix determinánsa – *szint: 3*). Olvassunk be fájlból egy $N \times N$ méretű mátrixot, ellenőrizzük le, hogy alsó vagy felső háromszögmátrixában mindenütt 0 elem szerepel-e! Ez esetben számoljuk ki a mátrix determinánsának értékét! Ellenkező esetben jelezzük a képernyőn, hogy a speciális feltételrendszer nem teljesül, így nem tudunk determinánsértéket meghatározni ezzel a módszerrel!

Magyarázat: Egyetlen menetben eldönthetjük, hogy a mátrix alsó vagy felső háromszögmátrixában csupa nulla érték szerepel-e. Ugyanis egy *i,j* indexű cella az alsó háromszögbe esik, ha $i < j$, főátlóba, ha $a = j$ teljesül, és felső háromszögbe, ha $i > j$. A vizsgálat és a determináns számítása a 11.27. és a 11.28. forráskódokban található.

11.12. forráskód. A vizsgálat kódja

```
int N = m.GetLength(0);
bool also = true, felso = true;
for(int i=0;i<N;i++)
{
   for (int j = 0; j < N; j++)
   {
      if (i < j && m[i, j] != 0) also = false;
      if (i > j && m[i, j] != 0) felso = false;
   }
}
```

11.13. forráskód. A determináns kiszámítása speciális háromszögmátrix esetben

```
if (also || felso)
{
   double det = 0;
   for (int i = 0; i < N; i++)
   {
      det = det * m[i, i];
   }
}
```

Bevezető információk: Speciális felépítésű a mátrix akkor, ha az első sorban mindenütt az 1 érték szerepel, a második sorában valamilyen számértékek (a, b, c, d, …), a harmadik sorában rendre ezen értékek négyzetei (a^2, b^2, c^2, d^2, …), a negyedik sorában a harmadik hatványok (a^3, b^3, c^3, d^3, …), és így tovább. Ha a mátrixunk szerkezete megfelel a leírtaknak, akkor a determináns értéke:

$$detA = a * b * c * d * \dots .$$

Ezt az értéket nevezzük ez esetben *Vandermonde-determinánsnak*, és belátható, hogy értéke ez esetben egyenlő a mátrix tényleges determinánsának értékével.

11.12. feladat (Vandermonde-determináns – *szint: 4*). Olvassunk be fájlból egy $N \times M$ méretű mátrixot, ellenőrizzük le, hogy a szerkezete eleget tesz-e a *Vandermonde-determináns* számítási módszere alkalmazhatóságának! Ha rendben van a mátrix, akkor számoljuk ki a mátrix determinánsát a Vandermonde számítási módszerével!

Magyarázat: Elsősorban ellenőrizzük le a Vandermonde-felépítést (11.29. forráskód). A továbbiakban kiszámíthatjuk a determinánst (11.30. forráskód).

11.14. forráskód. A Vandermonde felépítés ellenőrzése

```
static bool vandermonde_e(int[,] m)
{
   int N = m.GetLength(0);
   int M = m.GetLength(1);
   // 0. sor ellenorzese
   for (int j = 0; j < M; j++)
      if (m[0, j] != 1) return false;
   // 1. sor biztosan rendben
   // 2. sortol kezdodo sorok ellenorzse
   for (int i = 2; i < N; i++)
   {
      for (int j = 0; j < M; j++)
         if (m[i, j] != m[1, j] * m[i - 1, j]) return false;
   }
   // minden rendben volt
   return true;
}
```

11.15. forráskód. A Vandermonde determináns kiszámítása

```
static double vandermonde_det(int[,] m)
{
   double ret = 1.0;
   int M = m.GetLength(1);
   for (int j = 0; j < M; j++)
      ret = ret * m[1, j];
   return ret;
}
```

Bevezető információk: Egy elemekkel feltöltött $N \times N$ mátrix *lineárisan összefüggő*, ha van benne olyan sor (vagy oszlop), mely más sorok (vagy oszlopok) szorzatainak összegeként (lineáris kombinációjaként) előállítható. Ez legegyszerűbb esetben két egyforma értékekkel rendelkező sor vagy oszlop formájában jelentkezik, vagy olyan két sort (oszlopot) jelent, ahol az értékek nem pontosan egyenlők, de egy $\exists c \in \mathbb{R}$ konstans, hogy az egyik sorbeli értékek a másik sorbeli értékek c-szeresei. Vannak bonyolultabb esetek is, amikor valamely sor előállításában több más sor vesz részt (pl. a k. sor a i. sor c_i-szeres és a j. sor c_j-szeres értékeinek összege). Ezen felül tudjuk, hogy a lineárisan összefüggő mátrixok determinánsa nulla.

11.13. feladat (Lineárisan összefüggő mátrix – *szint: 3*). A mátrix értékeit olvassuk be fájlból, majd a determináns segítségével ellenőrizzük, hogy a

mátrix lineárisan összefüggő-e! Amennyiben igen, próbáljuk megkeresni, egyszerű lineáris függőségről van-e szó! Próbáljuk meg, találhatunk-e olyan sort (vagy oszlopot), amelynél a c konstans szorzó létezik! Ez esetben adjuk meg, melyik az a két sor (vagy oszlop), és mennyi a c értéke!

Magyarázat: A mátrix determinánsát a kifejtési tétel értelmében a 11.11-es feladatban leírtak szerint ki lehet számolni. Ha az nem nulla, úgy a mátrix nem lineárisan összefüggő, a további vizsgálódásra nincs szükség.

Ha a determináns nulla, úgy a lineáris összefüggés fennáll, csak nem tudni, hogy egyszerű vagy bonyolultabb esetről van-e szó. Feladatunk az egyszerű összefüggés vizsgálata. Minden sort minden más sorral, minden oszlopot minden más oszloppal össze kell vetnünk, hogy a közöttük esetleg fennálló lineáris összefüggést ellenőrizzük. Amennyiben a kiválasztott két sor (vagy oszlop) között létezik az említett *c* konstans, úgy azt a 11.8. feladatban leírtak szerint kaphatjuk meg.

A fejezet forráskódjai

11.16. forráskód. Amőbanyertesek megszámolása

```
odb = 0;
xdb = 0;
for(int i=0;i<N;i++)
{
for(int j=0;j<N;j++)
{
  // x-t tartalmazo cella
  if (m[i,j]==1)
  {
   if (sorban5(i,j))   xdb ++;
   if (oszlopban5(i,j)) xdb ++;
   if (atlosan5(i,j))  xdb ++;
  // o-t tartalmazo cella
  if (m[i,j]==2)
  {
   if (sorban5(i,j))   odb ++;
   if (oszlopban5(i,j)) odb ++;
   if (atlosan5(i,j))  odb ++;
  }
}
}
```

11.17. forráskód. A maximumkeresés vázlata

```
ahmx=2;
ahmy=1;
for(int i=1;i<N;i++)
for(int j=0;j<i;j++)
{
  if (m[ahmy,ahmx]<m[i,j])
  {
     ahmx = i;
     ahmy = j;
  }
}
```

11.18. forráskód. A nem nulla elemek megszámolása

```
foatlo_db=0;
egyeb_db=0;
for(int i=0;i<N;i++)
{
 for(int j=0;j<N;j++)
 {
   if (m[i,j]>0 && i==j) foatlo_db++;
   else if (m[i,j]>0 && i!=j) egyeb_db++;
 }
}
```

11.19. forráskód. A vizsgálat nem optimalizált változata

```
szimmetrikus = true;
ferdeszimmetrikus = true;

for(int i=0;i<N;i++)
{
 for(int j=0;j<N;j++)
 {
   if (m[i,j]!=m[j,i]) szimmetrikus=false;
   if (m[i,j]!=-m[j,i]) ferdeszimmetrikus=false;
 }
}
```

11.20. forráskód. A mátrix rangjának kalkulálása

```
int rang = 0;
for (int i = 0; i < N; i++)
{
   bool fuggetlen = true;
   int r_i = 0;
   for (j = 0; j < N; j++)
      if (i != j && lin_fuggetlen(i, j)) r_i++;
   if (r_i > rang) rang = r_i;
}
```

11.21. forráskód. A lineárisan függetlenség vizsgálata

```
static bool lin_fuggetlen(int i, int j)
{
 if (nullvektor(i)) return false;
 if (nullvektor(j)) return false;
 int k = elso_nem_nulla(j);
 double c = (double)m[i,k]/m[j,k];
 if (mind_nulla(i,j,c)) return true;
 else return false;
}
```

11.22. forráskód. A mátrix *i*. sora nullvektor-e

```
static bool nullvektor(int i)
{
 for(int j=0;j<K;j++)
 {
  if (m[i,j]!=0) return false;
 }
 return true;
}
```

11.23. forráskód. A mátrix *i*. sorának első nem 0 elemének oszlopindexe

```
static bool elso_nem_nulla(int i)
{
 for(int j=0;j<K;j++)
 {
   if (m[i,j]!=0) return j;
 }
 return 0;
}
```

11.24. forráskód. A mátrix *i*. és *j*. sorának *C* szerinti lineáris kombinációja nullvektor-e

```
static bool mind_nulla(int i, int j, double C)
{
 for(int j=0;j<K;j++)
 {
   if (m[i,k]+C*m[j,k]!=0) return false;
 }
 return true;
}
```

11.25. forráskód. Az adjungált mátrix előállítása

```
static int[,] adjungalt(int[,] m, int i, int j)
{
   int N = m.GetLength(0);
   int M = m.GetLength(1);
   int[,] ret = new int[N - 1, M - 1];
   int p = 0;
   for (int k = 0; k < N; k++)
   {
      if (k != i)
      {
         int q = 0;
         for (int l = 0; l < M; l++)
         {
            if (l != j)
            {
               ret[p, q] = m[k, l];
               q++;
            }

         }
         p++;
      }

   }
   return ret;
}
```

11.26. forráskód. A determináns kiszámítása

```
static double determinans(int[,] m)
{
   int N = m.GetLength(0);
   if (N == 1) return m[0, 0];
   double ret = 0.0;
   int elojel = 1;
   for (int i = 0; i < N; i++)
   {
      int[,] adj = adjungalt(m, 0, i);
      double d = determinans(adj);
      ret = ret + elojel*m[0, i] * d;
      elojel = -elojel;
   }
   return ret;
}
```

11.27. forráskód. A vizsgálat kódja

```
int N = m.GetLength(0);
bool also = true, felso = true;
for(int i=0;i<N;i++)
{
   for (int j = 0; j < N; j++)
   {
      if (i < j && m[i, j] != 0) also = false;
      if (i > j && m[i, j] != 0) felso = false;
   }
}
```

11.28. forráskód. A determináns kiszámítása speciális háromszögmátrix esetében

```
if (also || felso)
{
   double det = 0;
   for (int i = 0; i < N; i++)
   {
      det = det * m[i, i];
   }
}
```

11.29. forráskód. A Vandermonde-felépítés ellenőrzése

```
static bool vandermonde_e(int[,] m)
{
   int N = m.GetLength(0);
   int M = m.GetLength(1);
   // 0. sor ellenorzese
   for (int j = 0; j < M; j++)
      if (m[0, j] != 1) return false;
   // 1. sor biztosan rendben
   // 2. sortol kezdodo sorok ellenorzse
   for (int i = 2; i < N; i++)
   {
      for (int j = 0; j < M; j++)
         if (m[i, j] != m[1, j] * m[i - 1, j]) return false;
   }
   // minden rendben volt
   return true;
}
```

11.30. forráskód. A Vandermonde-determináns kiszámítása

```
static double vandermonde_det(int[,] m)
{
   double ret = 1.0;
   int M = m.GetLength(1);
   for (int j = 0; j < M; j++)
      ret = ret * m[1, j];
   return ret;
}
```

12. fejezet - Transzformációs mátrixok (szerző: Hernyák Zoltán)

A sík pontjait a hagyományos (x,y) koordinátapár helyett (x1,x2,x3) számhármassal írjuk le (homogén koordináták), ahol nem lehet mindhárom (x1,x2,x3 is) szám egyszerre 0 értékű.

A homogén koordináták esetén használt 3 elemű vektor által leírt síkbeli koordinátával kapcsolatosan többféle transzformációt alkalmazhatunk, például:

- eltolás valamilyen dx, dy értékkel,
- origó körüli elforgatás valamely α szöggel,
- tükrözés az x vagy y tengelyekre,
- ezek valamilyen lineáris kombinációja.

A transzformációk mindegyike leírható egy, az adott transzformációhoz konstruált speciális felépítésű 3×3 mátrixszal. A pont új koordinátáit az eredeti pontkoordinátákat leíró vektor és a transzformációs mátrix szorzata fogja generálni.

Az alábbi feladatok mindegyike igazából vektor–mátrix vagy mátrix–mátrix szorzásra visszavezethető feladat, mely a 11. fejezetben tárgyalt szorzó algoritmusokkal megoldható, így a feladatokhoz különösebb segítséget most kivételesen nem adunk.

Bevezető információk: A pont eltolása valamilyen (dx, dy) vektorral:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{bmatrix}$$

12.1. feladat (Eltolás számítása – *szint: 3*). A (3,2,0) homogén koordinátájú pontot toljunk el a síkban (-2,+3) vektorral! Számoljuk ki a pont eltolás utáni koordinátáit az eltolás mátrixával való szorzás révén!

Bevezető információk: A pont elforgatása az origó körül egy α szöggel:

$$\begin{bmatrix} cos\ \alpha & sin\ \alpha & 0 \\ -sin\ \alpha & cos\ \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

12.2. feladat (Elforgatás számítása – *szint: 3*). A (4,1,2) homogén koordinátájú pontot forgassuk el 45 fokkal az origó körül! Számoljuk ki a pont elforgatott képének koordinátáit az eltolás mátrixával való szorzás révén!

Bevezető információk: Amennyiben valamely pontot kívánunk tükrözni az X tengelyre, úgy az alábbi felépítésű mátrixszal való szorzásra van szükség:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Az Y tengelyre tükrözés esetén a következő mátrixra lesz szükségünk:

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

12.3. feladat (Tükrözések – *szint: 2*). Számoljuk ki a (3,4,1) homogén koordinátájú pont X majd Y tengelyre tükrözés utáni koordinátáit! Vessük össze, hogy egyezik-e az eredmény az origó körüli 180 fokos forgatás után kapott koordinátákkal!

Bevezető információk: Az *sx*, *sy* értékekkel történő skálázást az alábbi mátrix írja le:

$$\begin{bmatrix} \mathrm{sx} & 0 & 0 \\ 0 & \mathrm{sy} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

12.4. feladat (Skálázás – *szint: 2*). Számoljuk ki az (5,3,2) homogén koordinátájú pont 3.2 és 4.5 skálaértékű transzformációjának eredményét!

13. fejezet - A mágikus és bűvös négyzetek (szerző: Hernyák Zoltán)

Tartalom

A fejezet forráskódjai

Bevezető információk: Egy *A* mátrix soraira nézve sztochasztikus, ha elemei nemnegatívak, és minden sorösszege 1, és oszlopaira nézve sztochasztikus, ha elemei nemnegatívak, és minden oszlopösszege 1. Ha soraira és oszlopaira nézve is sztochasztikus, akkor kétszeresen (duplán) sztochasztikusnak nevezzük.

13.1. feladat (Sztochasztikus mátrix – *szint: 2*). Egy text fájlban szereplő (tört számokat tároló) $N \times M$ mátrix elemeit olvassuk be, majd dönstük el, hogy a mátrix duplán sztochasztikus-e!

Magyarázat: Ezt egy egyszerű függvény képes eldönteni (13.41. forráskód). A „minden szám nemnegatív” vizsgálatot érdemes belefoglalni valamelyik sor- vagy oszlopösszeg- ellenőrző lépésbe, mivel azok szkennelik a mátrix minden elemét, illetve kis ügyességgel egyetlen mátrixbejárással elvégezhető a teljes ellenőrzés (13.42. forráskód).

13.1. forráskód. Duplán sztochasztikusság ellenőrzése klasszikus megoldással

```
static bool duplan_sztochasztikus(int[,] m)
{
   int N = m.GetLength(0);
   int M = m.GetLength(1);
   // minden eleme nemnegatív ?
   for (int i = 0; i < N; i++)
      for (int j = 0; j < M; j++)
         if (m[i, j] < 0) return false;
   // sorosszeg mindenütt 1.0 ?
   for (int i = 0; i < N; i++)
   {
      double sum = 0.0;
      for (int j = 0; j < M; j++)
         sum = sum + m[i, j];
      if (sum != 1.0) return false;
   }
   // oszlop osszegek mindenütt 1.0 ?
   for (int j = 0; j < M; j++)
   {
      double sum = 0.0;
      for (int i = 0; i < N; i++)
         sum = sum + m[i, j];
      if (sum != 1.0) return false;
   }
   // minden rendben
   return true;
}
```

13.2. forráskód. Duplán sztochasztikusság ellenőrzése optimalizált módon

```
static bool duplan_sztochasztikus_opt(int[,] m)
{
   int N = m.GetLength(0);
   int M = m.GetLength(1);
   double[] sorok = new double[N];
   double[] oszlopok = new double[M];
   // matrix bejarasa
   for (int i = 0; i < N; i++)
      for (int j = 0; j < M; j++)
      {
```

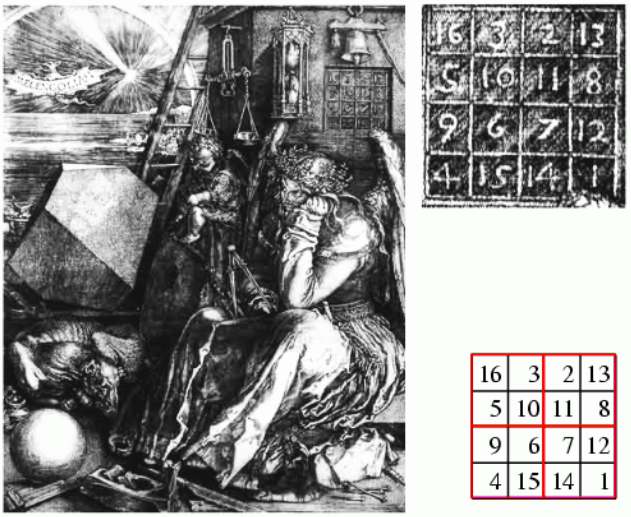
```
            if (m[i, j] < 0) return false;
            sorok[i] = sorok[i] + m[i, j];
            oszlopok[j] = oszlopok[j] + m[i, j];
         }
      // ellenorzes
      for (int i = 0; i < N; i++)
      if (sorok[i]!=1.0) return false;
      for (int j = 0; j < M; j++)
         if (oszlopok[j] != 1.0) return false;
      // minden rendben
      return true;
   }
```

Bevezető információk: A duplán sztochasztikus mátrixokhoz hasonlóak a *mágikus négyzetek*. Ezek olyan mátrixok, melyek egész számokat tartalmaznak, minden szám egyedi (nem ismétlődik a mátrixban), és soraikban, oszlopaikban, valamint az átlókban szereplő számok összege egyenlő egymással. Ezt az értéket nevezzük mágikus értéknek, vagy *kulcsértéknek*[1].

A mágikus négyzetek több ezer éve ismertek, a középkorban talizmánként is használták őket. Egyik híres felbukkanási helye Albrecht Dürer (1471–1528) német festő és grafikus *Melankólia* c. metszetének (lásd a 13.1. ábra) jobb felső sarka, ahol egy 4×4 méretű mágikus négyzet[2] látható. A négyzet alsó sorában középen szereplő két érték összeolvasva 1514, mely a kép keletkezésének éve is egyben.

A mágikus négyzeteket már a kínaiak is ismerték, egyik legrégebbi felbukkanása a *változások könyve* jóskönyv, mely i. e. 3000 körül készülhetett.

13.1. ábra. Albrecht Dürer – Melankólia



16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

13.2. feladat (Mágikus négyzet-e – *szint: 2*). Egy fájlban szereplő $N \times N$ méretű mátrixról döntsük el, hogy *mágikus négyzet-e*! Ha igen, adjuk meg a mágikus mátrix kulcsértékét!

Magyarázat: A 13.2. ábrán látható egy Benjamin Franklin által készített 8×8 méretű mágikus négyzet[3], ahol a sorok és oszlopok összege 260. A 13.1 feladatban elkészített ellenőrző függvény módosításokkal alkalmas a mágikus négyzet tulajdonság ellenőrzésére is. A különbség nemcsak az, hogy nem 1.0-nak kell lenni az összegeknek, de egymással is egyezőknek. Ha minden sorösszeg egyezik az első sor összegével, valamint minden oszlopösszeg egyezik az első oszlop összegével, akkor ez a tulajdonság már majdnem rendben is van. De meg kell még vizsgálni, hogy a sorok és oszlopok összege egymással is egyenlő-e! Ezenfelül ugyanezen összegnek kell szerepelni a két főátlóbeli számok összegeiként is. Nem szabad elfelejtkezni a számok egyediségének ellenőrzéséről sem! Mindezek a 13.43. ... 13.47. forráskódokban találhatók.

13.2. ábra. Franklin 8×8 méretű mágikus négyzete

52	**61**	**4**	**13**	**20**	**29**	**36**	**45**	260
14	**3**	**62**	**51**	**46**	**35**	**30**	**19**	260
53	**60**	**5**	**12**	**21**	**28**	**37**	**44**	260
11	**6**	**59**	**54**	**43**	**38**	**27**	**22**	260
55	**58**	**7**	**10**	**23**	**26**	**39**	**42**	260
9	**8**	**57**	**56**	**41**	**40**	**25**	**24**	260
50	**63**	**2**	**15**	**18**	**31**	**34**	**47**	260
16	**1**	**64**	**49**	**48**	**33**	**32**	**17**	260
260	260	260	260	260	260	260	260	

13.3. forráskód. Bűvös négyzet – „buvos_negyzet_e”

```
static bool magikus_negyzet_e(int[,] m)
{
   int N = m.GetLength(0);
   int[] sorok = new int[N];
   int[] oszlopok = new int[N];
   osszegekGen(m, sorok, oszlopok);
   // sorok, oszlopok osszegei
   if (mindEgyenlo(sorok) == false ||
      mindEgyenlo(oszlopok) == false ||
      sorok[0] != oszlopok[0]) return false;
   // atlok osszegei
   int atlo1, atlo2;
```

```
      atlokGen(m, out atlo1, out atlo2);
      if (atlo1 != atlo2 || atlo1 != sorok[0])
         return false;
      // egyediseg
      for (int i = 0; i < N; i++)
         for (int j = 0; j < N; j++)
            if (megszamol(m, m[i, j]) != 1)
               return false;
      // minden rendben
      return true;
   }
```

13.4. forráskód. Bűvös négyzet – „osszegekGen”

```
static void osszegekGen(int[,] m, int[] sorok, int[] oszlopok)
{
 for (int i = 0; i < sorok.Length; i++)
   for (int j = 0; j < oszlopok.Length; j++)
   {
   sorok[i] = sorok[i] + m[i, j];
   oszlopok[j] = oszlopok[j] + m[i, j];
   }
}
```

13.5. forráskód. Bűvös négyzet – „mindEgyenlo”

```
   static bool mindEgyenlo(int[] l)
   {
      int x = l[0];
      foreach (int a in l)
         if (x != a) return false;
      return true;
   }
```

13.6. forráskód. Bűvös négyzet – „atlokGen”

```
   static void atlokGen(int[,] m, out int a, out int b)
   {
      a = 0; b = 0;
      int N = m.GetLength(0);
      for (int i = 0; i < N; i++)
      {
         a = a + m[i, i];
         b = b + m[i, N - i - 1];
      }
   }
```

13.7. forráskód. Bűvös négyzet – „megszamol”

```
   static int megszamol(int[,] m, int x)
   {
      int N = m.GetLength(0);
      int db = 0;
      for (int i = 0; i < N; i++)
         for (int j = 0; j < N; j++)
            if (m[i, j] == x) db++;
      return db;
   }
```

Bevezető információk: Az olyan mágikus négyzeteket, melyekben minden érték prímszám, prím mágikus négyzeteknek nevezzük (lásd pl. a 13.3. ábra).

13.3. ábra. Prímszámokból felépített 4×4 méretű mágikus négyzet

37	**83**	**97**	**41**	258
53	**61**	**71**	**73**	258
89	**67**	**59**	**43**	258
79	**47**	**31**	**101**	258
258	258	258	258	

13.3. feladat (Prím mágikus négyzet-e – *szint: 3*). Feladat: egy fájlból beolvasott $N \times N$ méretű mátrixról döntsük el, hogy bűvös négyzet-e, és ha igen, akkor prímekből felépített mágikus négyzet-e! Soroljuk fel tételesen, melyik cellában szereplő mely értékek nem prímek, amelyek elrontják ezt a tulajdonságot! A megoldás során az 1 értéket kivételesen kell kezelni. Ha szerepel valamelyik cellában, de mindegyik másik cellaérték prím, akkor az még elfogadható prím mágikus négyzet kategóriának.

Magyarázat: A prímszám eldöntésére több módszer is létezik, jelen probléma esetén majdnem mindegy, melyik módszert választjuk. Egy „prímszám-e” bevizsgáló függvény megírása után a probléma kezelhető szintre redukálódik: a 13.2 feladatban leírt ellenőrzést esetünkben csak azzal kell kiegészíteni, hogy a mátrix minden egyes számáról el kell dönteni, hogy prímszám-e vagy sem.

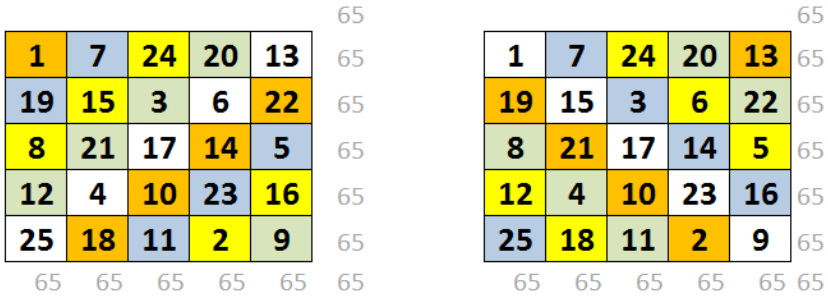
13.8. forráskód. A mátrix értékeinek ellenőrzése

```
bool mindegyikPrim = true;
for(int i=0;i<N && mindegyikPrim;i++)
{
 for(int j=0;j<M && mindegyikPrim;j++)
 {
   if (prim-e(m[i,j]==false)
      mindegyikPrim = false;
 }
```

```
}
```

Bevezető információk: A *pánmágikus* négyzetek nem egyszerű mágikus négyzetek, hanem egyéb jellemzőkkel is bírnak. Nemcsak a sorokban és oszlopokban szereplő értékek összege egyenlő, hanem a főátlókban lévő számok összege is, valamint az ún. *törtátlókban* is. A 13.4. ábrán látható, mit értünk törtátlón. Egy 5×5 méretű mátrixnál berajzoltuk ezeket is. A 13.5. ábrán újabb pánmágikus mátrixokat mutatunk be.

13.4. ábra. Pánmágikus mátrixok törtátlói



13.5. ábra. Újabb lehetséges pánmágikus mátrixok

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

4	5	10	15
14	11	8	1
7	2	13	12
9	16	3	6

13.4. feladat (Pánmágikus négyzet – *szint: 4*). A feladat: írjunk olyan programot, amely beolvas fájlból egy $N \times N$ méretű mátrixot, és eldönti, hogy pánmágikus-e! A program jelenítse meg az egyes törtátlókat eltérő színnel, és adja meg az adott törtátlóban lévő értékek összegét (egy időben csak egy törtátlóra koncentrálva, vagyis az adott törtátló számai legyenek zöldek, minden más szám legyen szürke)!

A mátrix törtátlós színezésével foglalkozik a 13.49. forráskód (jobbra-le törtátlók), a másik (balra-le) törtátlós színes kiírásával a 13.50. forráskódban található egyfajta megoldás. A színeket egy vektorba helyezzük. Konzolos felületen limitált mennyiségű színt használhatunk fel, és érdemes úgy válogatni a színeket egymás mellé, hogy lényegesen elüssenek egymástól. Ezért a színek vektorát manuálisan töltöttük fel színekkel. A két kiírás között egyébként a lényeges különbség mindössze a külső *i* ciklusban lévő *szin* kiválasztás módja.

13.9. forráskód. Jobbra-le törtátlók

```
static void panmagikus_kiir_A(int[,] m)
{
   int N = m.GetLength(0);
   ConsoleColor[] color = new ConsoleColor[] { ConsoleColor.Red,
       ConsoleColor.Green, ConsoleColor.Yellow, ConsoleColor.Cyan,
       ConsoleColor.White, ConsoleColor.Magenta };
   //
   int szin = 0;
   for (int i = 0; i < N; i++)
   {
      szin = i;
      for (int j = 0; j < N; j++)
      {
         Console.ForegroundColor = color[szin];
         Console.Write("{0,4}", m[i, j]);
         szin++;
         if (szin >= N) szin = 0;
      }
      Console.WriteLine();
   }
}
```

13.10. forráskód. Balra-le törtátlók

```
static void panmagikus_kiir_A(int[,] m)
{
   int N = m.GetLength(0);
   ConsoleColor[] color = new ConsoleColor[] { ConsoleColor.Red,
       ConsoleColor.Green, ConsoleColor.Yellow, ConsoleColor.Cyan,
       ConsoleColor.White, ConsoleColor.Magenta };
   //
   int szin = 0;
   for (int i = 0; i < N; i++)
   {
      szin = i;
      for (int j = 0; j < N; j++)
      {
         Console.ForegroundColor = color[szin];
         Console.Write("{0,4}", m[i, j]);
         szin++;
         if (szin >= N) szin = 0;
      }
      Console.WriteLine();
   }
```

```
}
```

A színek váltogatása során ismertetett módszer egyúttal a törtátlók szummájának képzéséhez is szükséges. A 13.51. forráskódban a két törtátlós összegek képzése és összehasonlítása történik meg. Ne feledjük azonban, hogy ezenfelül szükséges még a sorok és az oszlopok összegeinek ellenőrzése is (valamint a sorok és oszlopok összegeinek is egyezniük kell a törtátlók összegeivel)!

13.11. forráskód. Törtátlókban található összegek képzése és összehasonlítása

```
static bool panmagikus_ell(int[,] m)
{
   int N = m.GetLength(0);
   int[] sum1 = new int[N];
   int[] sum2 = new int[N];
   //
   int i1, i2;
   for (int i = 0; i < N; i++)
   {
      i1 = i;
      i2 = N - i - 1;
      for (int j = 0; j < N; j++)
      {
         sum1[i1] = sum1[i1] + m[i, j];
         sum2[i2] = sum2[i2] + m[i, j];
         i1++;
         if (i1>= N) i1= 0;
         i2++;
         if (i2 >= N) i2 = 0;
      }
   }
   //
   for (int i = 1; i < N; i++)
      if (sum1[i] != sum1[0])
         return false;
   for (int i = 1; i < N; i++)
      if (sum2[i] != sum2[0])
         return false;
   if (sum1[0] != sum2[0])
      return false;
   // minden ok
   return true;
}
```

Bevezető információk: Az ördögkeretek[4] olyan (nagyobb méretű) mágikus négyzetek, melyek külső keretét eltávolítva szintén mágikus négyzetet kapunk – tehát mágikus négyzetek vannak egymásba ágyazva. Értelemszerűen ezen kisebb méretű mágikus négyzet kulcsértéke is kisebb, mivel a keretet alkotó értékek már nem szerepelnek a sorok és oszlopok összegszámításában. Érdekesebb esetben ez az egymásba ágyazás több szinten is előfordulhat, mígnem egy olyan belső mátrixhoz jutunk el, amire már nem teljesül, hogy ő is egy önálló mágikus négyzet. A 13.6. ábrán egy 12-ed rendű ördögkeret látható.

13.6. ábra. 12-ed rendű ördögkeret

1	143	142	4	5	139	138	8	9	135	134	12	870
13	23	121	120	119	27	29	31	113	112	30	132	870
131	117	41	103	102	44	45	99	98	48	28	14	870
130	105	96	55	89	88	59	84	60	49	40	15	870
129	39	95	87	65	79	78	68	58	50	106	16	870
128	107	51	62	76	70	71	73	83	94	38	17	870
18	37	93	82	72	74	75	69	63	52	108	127	870
19	36	53	64	77	67	66	80	81	92	109	126	870
125	35	54	85	56	57	86	61	90	91	110	20	870
21	111	97	42	43	101	100	46	47	104	34	124	870
22	115	24	25	26	118	116	114	32	33	122	123	870
133	2	3	141	140	6	7	137	136	10	11	144	870
870	870	870	870	870	870	870	870	870	870	870	870	

13.5. feladat (Ördögkeretek – *szint: 4*). Írjunk olyan programot, amely beolvas egy fájlból egy $N \times N$ méretű mátrixot, és meghatározza, hogy milyen mélységben tartalmaz mágikus négyzeteket egymásba ágyazva! Ha ez a szám 0, akkor már a kiinduló mátrix sem volt mágikus négyzet.

Magyarázat: Ehhez 13.2 feladatban leírt mágikus négyzet ellenőrzése módszert kell kiterjeszteni, hogy ne a teljes mátrixra, csak annak egy részére terjedjen ki az ellenőrzés. Ez a módosítás nemcsak a fő ellenőrző függvényt (*buvos_negyzet_e*) érinti, hanem a belőle hívott segédfüggvényeket is.

13.12. forráskód. Bűvös négyzet – „buvos_negyzet_e_2”

```
static bool magikus_negyzet_e_2(int[,] m,int eltolas)
{
   int N = m.GetLength(0) - 2 * eltolas;
   int[] sorok = new int[N];
   int[] oszlopok = new int[N];
   osszegekGen2(m, sorok, oszlopok, eltolas);
   // sorok, oszlopok osszegei
   if (mindEgyenlo(sorok) == false ||
      mindEgyenlo(oszlopok) == false ||
      sorok[0] != oszlopok[0]) return false;
   // atlok osszegei
   int atlo1, atlo2;
   atlokGen2(m, out atlo1, out atlo2,eltolas);
   if (atlo1 != atlo2 || atlo1 != sorok[0])
      return false;
   // egyediseg
   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
```

```
            if (megszamol2(m, m[i, j],eltolas) != 1)
               return false;
      // minden rendben
      return true;
   }
```

13.13. forráskód. Bűvös négyzet – „osszegekGen2”

```
static void osszegekGen2(int[,] m,int[] sorok,int[] oszlopok,int eltolas)
{
  for (int i = 0; i < sorok.Length; i++)
     for (int j = 0; j < oszlopok.Length; j++)
     {
         sorok[i] = sorok[i] + m[i + eltolas, j + eltolas];
         oszlopok[j] = oszlopok[j] + m[i + eltolas, j + eltolas];
     }
}
```

13.14. forráskód. Bűvös négyzet – „mindEgyenlo”

```
static bool mindEgyenlo(int[] l)
{
   int x = l[0];
   foreach (int a in l)
      if (x != a) return false;
   return true;
}
```

13.15. forráskód. Bűvös négyzet – „atlokGen2”

```
static void atlokGen2(int[,] m, out int a, out int b, int eltolas)
{
   a = 0; b = 0;
   int N = m.GetLength(0);
   for (int i = 0; i < N; i++)
   {
      a = a + m[i + eltolas, i + eltolas];
      b = b + m[i + eltolas, N - i - 1 - +eltolas];
   }
}
```

13.16. forráskód. Bűvös négyzet – „megszamol2”

```
static int megszamol2(int[,] m, int x, int eltolas)
{
   int N = m.GetLength(0)-eltolas;
   int db = 0;
   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
        if (m[i+eltolas, j+eltolas] == x) db++;
   return db;
}
```

Bevezető információk: Az olyan $N \times N$ méretű mátrixok, amelyekben minden szám szerepel $[1, N^2]$ intervallumból, s melyeknél a sorok, az oszlopok és az átlókban szereplő összegek különbözőek: *antimágikus négyzetnek* nevezzük. Bizonyítható, hogy nem létezik $1 \times 1, 2 \times 2, 3 \times 3$ méretű anti mágikus négyzet.

13.6. feladat (Antimágikus négyzet ellenőrzése – *szint: 2*). Egy $N \times N$ méretű kitöltött mátrixot ellenőrizzünk le, hogy *antimágikus négyzet-e*! Az ellenőrzés után jelenítsük meg a mátrixot a képernyőn, minden sorhoz és oszlophoz jelenítsük meg az összegeket, a főátló és a mellékátló összegét is! Az ellenőrzés eredményét írjuk vörös színnel, ha nem felelt meg, és zölddel, ha megfelelt!

Magyarázat: A főprogramban (a 13.57. forráskód) a mátrixot az alapadatokal történő feltöltés után megjelenítjük a képernyőn. A 13.59. forráskódban egy intelligens megjelenítő függvényt készítettünk, amely maga számolja ki a sor- és oszlopösszegeket, valamint az átlók összegét (semmit sem bízván a véletlenre). A sorok összegét kiírás közben számolja a *sum* változóba, és minden sor kiírásának végén azt meg is jeleníti. Az oszlopösszegeket a *oszlopok* vektorban számolja, mert az csak a legalsó sor kiírása után lesz látható. A főátlóbeli összeget a jobb sarokban jeleníti meg, a *foatlo* változó alapján. A mellékátló összegét az alsó sorban, az oszlopátlók előtt írja ki. A vizsgálatot a 13.58. forráskódban szereplő *anti_buvos_negyzet_e* függvény végzi. A sorok összege N, az oszlopok összege is N, a főátlóbeli és a mellékátlóbeli elemek összege 2 darab szám, így összesen 2 * N + 2 összeget kell kiszámítani, ezért készül az *osszegek* vektor ezzel a mérettel el. Az első N elemében szerepelnek majd a sorok összegei, a további N elemében az oszlopok, az utolsó előtti a főátló, az utolsó a mellékátló összege. A vektor feltöltése után ellenőrizzük, hogy van-e az összegek között két egyforma. Ha idáig minden rendben, akkor még azt is ellenőrizni kell, hogy minden szám szerepel-e az $[1,2^N]$ intervallumból, mindegyik pontosan egyszer.

13.7. ábra. A program outputja

13.17. forráskód. Anti bűvös négyzet-e – teszt főprogram

```
static void Main()
{
   const int N = 5;
   int[,] m = new int[N, N];
   // kezdo feltoltes
```

```
   int a = 1;
   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
      {
         m[i, j] = a;
         a++;
      }
   //
   Magikus_Kiiras_2(m);
   //
   Console.SetCursorPosition(0, N + 2);
   if (anti_magikus_negyzet_e(m) == false)
   {
      Console.ForegroundColor = ConsoleColor.Red;
      Console.WriteLine("NEM ANTI-MAGIKUS NEGYZET");
   }
   else
   {
      Console.ForegroundColor = ConsoleColor.Green;
      Console.WriteLine("IGENIS ANTI-MAGIKUS NEGYZET");
   }
   Console.ReadLine();
}
```

13.18. forráskód. Anti bűvös négyzet-e – az ellenőrző függvény

```
static bool anti_magikus_negyzet_e(int[,] m)
{
 int N = m.GetLength(0);
 int[] osszegek = new int[2*N+2];
 // az osszegek kepzese
 for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
   {
    // i. sor osszege
    osszegek[i]  = osszegek[i] + m[i, j];
    // j. oszlop osszege
    osszegek[N+j] = osszegek[i] + m[i, j];
    // foatlo osszege
    if (i==j) osszegek[2 * N] = osszegek[2*N] + m[i, j];
    // mellekatlo osszege
    if (i == N-j-1) osszegek[2 * N+1] = osszegek[2 * N+1] + m[i, j];
   }

 // van-e ket egyforma osszeg ?
 for (int i = 0; i < osszegek.Length; i++)
   for (int j = i + 1; j < osszegek.Length; j++)
    if (osszegek[i] == osszegek[j])
      return false;
 // minden szam szerepel?
 for (int k = 1; k <= N * N; k++)
 {
   int db = 0;
   for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
      if (m[i, j] == k) db++;
   if (db != 1) return false;
 }
 // minden ok
 return true;
}
```

13.19. forráskód. Anti bűvös négyzet-e – mátrix megjelenítés

```
static void Magikus_Kiiras(int[,] m)
{
   int N = m.GetLength(0);
   int[] oszlopok = new int[N];
   int foatlo = 0;
   int mellekatlo = 0;
   //
   //
   for (int i = 0; i < N; i++)
   {
      Console.ForegroundColor = ConsoleColor.Yellow;
      int sum = 0;
      Console.Write("  ");
      for (int j = 0; j < N; j++)
      {
         Console.Write("{0,3} ", m[i, j]);
         //
         sum = sum + m[i, j];
         oszlopok[j] = oszlopok[j] +m[i, j];
         if (i==j) foatlo=foatlo+m[i,j];
         if (i == N - j - 1)
            mellekatlo = mellekatlo + m[i, j];
      }
      //
      Console.ForegroundColor = ConsoleColor.Cyan;
      Console.Write("{0,4} ", sum);
      Console.WriteLine();
   }
   //
   Console.ForegroundColor = ConsoleColor.Green;
   Console.Write("{0,3} ", mellekatlo);
   Console.ForegroundColor = ConsoleColor.Cyan;
   for (int j = 0; j < N; j++)
      Console.Write("{0,3} ", oszlopok[j]);
   Console.ForegroundColor = ConsoleColor.Green;
   Console.Write("{0,3} ", foatlo);
   Console.ForegroundColor = ConsoleColor.Gray;
```

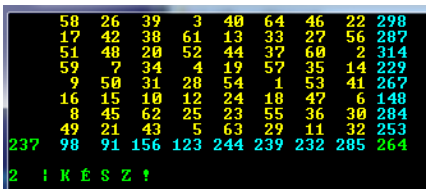
```
}
```

13.7. feladat (Antimágikus négyzet generálása – *szint: 2*). Generáljunk egy $N \times N$ méretű mátrixot oly módon, hogy a végeredménye antimágikus négyzet legyen!

Magyarázat: A generálás során először feltöltjük a mátrixot módszeresen $[1, N^2]$ közötti értékekkel. Majd kiszámítjuk a sorok, oszlopok, átlók összegeit az előző feladatban ismertetett módon egy 2 * N + 2 méretű vektorba. Megkeressük, melyik két összeg egyenlő egymással. Ha nincs egyenlőség, akkor készen vagyunk. Ha találunk egyenlőséget, akkor választunk egy cellát a problémás sorból/oszlopból/átlóból, valamint egy véletlen cellát a mátrix tetszőleges helyéről. A két cellában lévő értéket felcseréljük.

Ezt addig ismételjük, amíg már nem lesz egyenlőség sehol. A mátrix megjelenítésén is finomítottunk: amelyik két összeg egyenlő egymással, azokat piros színnel jelenítjük meg. Valamint kiírásra kerül az is, hogy hány cserét kellett elvégezni, hogy a kívánt eredményt elérjük. Ez általában kevés csere, mivel a mátrix kezdeti feltöltése majdnem megfelelő. Ezért azzal bonyolítottuk a megoldást, hogy a kezdeti feltöltés után sok cserét hajtottunk végre a mátrix cellái között, hogy egy *összekevert* kezdő állapotot elérjünk. A mátrix ezen állapota is elég kedvező az antimágikus négyzet kiindulási állapotához, mert ezek után sincs szükség sok cserére. Úgy tűnik, ilyen négyzetek előállítása könnyű. A 13.60. ... 13.65. közötti forráskódok fedik le a megoldást.

13.8. ábra. A program outputja



13.20. forráskód. Anti bűvös négyzet generálás – főprogram

```
static void Main()
{
   const int N = 8;
   int[,] m = new int[N, N];
   int[] osszegek = new int[2 * N + 2];
   kezdoFeltoltes(m);
   osszekeveres(m, N * N * 40);
   //
   ulong fdb = 0;
   while (true)
   {
      int i, j;
      osszegekSzamol(m, osszegek);
      if (egyformaIndexek(osszegek, out i, out j))
      {
         int a, b, c, d;
         valasztElem(i, out a, out b, N);
         c = rnd.Next(0, N);
         d = rnd.Next(0, N);
         // csere
         int x = m[a, b];
         m[a, b] = m[c, d];
         m[c, d] = x;
      }
      else break;
      // folyamat kozbeni kijelzes
      if (fdb % 100000 == 0)
      {
         Console.Clear();
         Magikus_Kiiras_2(m);
         Console.SetCursorPosition(0, N + 1);
         Console.Write(fdb);
      }
      fdb++;
   }
   //
   Console.Clear();
   Magikus_Kiiras_2(m);
   Console.SetCursorPosition(0, N + 2);
   Console.WriteLine("{0} | K É S Z ! ", fdb);
   Console.ReadLine();
}
```

13.21. forráskód. Anti bűvös négyzet generálás – az összegek számítása

```
static void osszegekSzamol(int[,] m, int[] osszegek)
{
   int N = m.GetLength(0);
   for (int i = 0; i < osszegek.Length; i++)
      osszegek[i] = 0;
   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
      {
         // i. sor osszege
         osszegek[i] = osszegek[i] + m[i, j];
         // j. oszlop osszege
         osszegek[N + j] = osszegek[i] + m[i, j];
         // foatlo osszege
         if (i == j)
            osszegek[2 * N] = osszegek[2 * N] + m[i, j];
         // mellekatlo osszege
         if (i == N - j - 1)
            osszegek[2 * N + 1] = osszegek[2 * N + 1] + m[i, j];
      }
}
```

13.22. forráskód. Anti bűvös négyzet generálás – mátrix megjelenítése v2

```
static void Magikus_Kiiras_2(int[,] m)
{
   int N = m.GetLength(0);
   int[] osszegek = new int[2 * N + 2];
   osszegekSzamol(m, osszegek);
   int x,y;
   egyformaIndexek(osszegek, out x, out y);

   //
   Console.ForegroundColor = ConsoleColor.Yellow;
   for (int i = 0; i < N; i++)
   {
      Console.Write("  ");
      for (int j = 0; j < N; j++)
      {
         Console.ForegroundColor = ConsoleColor.Yellow;
         Console.Write("{0,3} ", m[i, j]);
      }
      Console.WriteLine();
   }
   // sorok osszegei
   for(int i=0;i<N;i++)
   {
      if (i==x || i==y) Console.ForegroundColor = ConsoleColor.Red;
      else Console.ForegroundColor = ConsoleColor.Cyan;
      Console.SetCursorPosition(N*4+4,i);
      Console.Write("{0,3} ", osszegek[i]);
   }
   // oszlopok osszegei
   for (int i = 0; i < N; i++)
   {
      if (i+N == x || i+N == y)
        Console.ForegroundColor = ConsoleColor.Red;
      else
        Console.ForegroundColor = ConsoleColor.Cyan;
      Console.SetCursorPosition(4+i*4, N);
      Console.Write("{0,3} ", osszegek[i+N]);
   }
   // foatlo
   if (2*N == x || 2*N == y) Console.ForegroundColor = ConsoleColor.Red;
   else Console.ForegroundColor = ConsoleColor.Green;
   Console.SetCursorPosition(4 + N * 4, N);
   Console.Write("{0,3} ", osszegek[2*N]);
   // mellektalo
   if (2 * N+1 == x || 2 * N+1 == y)
      Console.ForegroundColor = ConsoleColor.Red;
   else
      Console.ForegroundColor = ConsoleColor.Green;
   Console.SetCursorPosition(0, N);
   Console.Write("{0,3} ", osszegek[2 * N+1]);
}
```

13.23. forráskód. Anti bűvös négyzet generálás – elem választása

```
static void valasztElem(int i, out int a, out int b, int N)
{
   // i egy sor indexe
   if (i < N)
   {
      a = i;
      b = rnd.Next(0, N);
      return;
   }
   // i egy oszlop indexe
   if (i < 2*N)
   {
      a = rnd.Next(0, N);
      b = i-N;
      return;
   }
   // i a foatlo
   if (i == 2 * N)
   {
      a = rnd.Next(0, N);
      b = a;
      return;
   }
   // i a mellekatlo
   a = rnd.Next(0, N);
   b = N - a - 1;
}
```

13.24. forráskód. Anti bűvös négyzet generálás – mátrix kezdőfeltöltése

```
static void kezdoFeltoltes(int[,] m)
{
   int N = m.GetLength(0);
   int k = 1;
   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
      {
         m[i, j] = k;
         k++;
      }
```

```
}
```

13.25. forráskód. Anti bűvös négyzet generálás – elem összekeverése

```
static void osszekeveres(int[,] m, int db)
{
   int N = m.GetLength(0);
   while (db > 0)
   {
      int x = rnd.Next(0, N);
      int y = rnd.Next(0, N);
      int v = rnd.Next(0, N);
      int w = rnd.Next(0, N);
      //
      int c = m[x, y];
      m[x, y] = m[v, w];
      m[v, w] = c;
      //
      db--;
   }
}
```

13.8. feladat (Antimágikus négyzet befejezése – *szint: 3*). Egy text fájlból olvassunk be egy $N \times N$ mátrixot, melynek bizonyos celláiban a 0 érték fog szerepelni! Ezen 0 értékeket tekintsük úgy, mintha a mátrix ezen a ponton még kitöltetlen lenne! Fejezzük be az antimágikus négyzetet oly módon, hogy a kitöltetlen cellákba $[1, N^2]$ közötti számokat helyezünk! Ügyeljünk arra, hogy a feladat esetleg megoldhatatlan! Ha ha a kitöltött cellákban ismétlődés szerepel, vagy van teljesen kitöltött sor, oszlop, átló, melyek valamelyikében a benne szereplő értékek összege megegyezik, akkor nem lehetséges a definíciónak megfelelni.

Magyarázat: A mátrix kitöltetlen celláiba eleve olyan értékeket kell elhelyezni az $[1, N^2]$ értékekből, amelyek még nem szerepeltek benne. A továbbiakban az előző feladatban ismertetett módon hajthatunk végre cseréket a mátrixon belül, csak arra kell ügyelnünk, hogy a kitöltött cellákat kihagyjuk. Ez azonban nem is olyan egyszerű. A *valasztElem* függvényt alaposan módosítani kell, ha például egy teljes kitöltött sorból kellene neki véletlen cellát választani, akkor az nem sikerülhet. Emiatt *Main* függvényben is alaposan át kell gyúrni ezt a részt.

Külön problémát okoz, hogyan különítjük el a fixen kitöltött cellákat a kitöltetlenektől. Ennek támogatásához bevezettünk egy *fix* nevű logikai mátrixot, melyben a benne lévő érték *true*, ha a cella fixen kitöltött, *false* ha szabadon módosítható. A mágikus négyzetet megjelenítő eljárást is módosítottuk, hogy a fix cellákat más színnel jelenítse meg, mint a szabad cellákat.

A program a fájl beolvasása után megjeleníti a kitöltött és kitöltetlen cellákat (a fix cellákat zöld alapon feketével). Majd kitölti a maradék cellákat a soron következő sorszámokkal, és ezt az állapotot is megjeleníti, végül megpróbálja megoldani a feladatot. A 13.74. és a 13.73. forráskódok olyan szinten fedik le a feladatot, hogy a megoldhatatlansági kérdéssel nem foglalkozunk.

- A *Main* függvény először beolvassa a fájl tartalmát, majd befejezi a kitöltést. A továbbiakban hasonlóan működik, mint az előző feladatban leírtak.
- A *beolvasas* függvény a leírtak szerint beolvassa a mátrix tartalmát a text fájlból, és közben párhuzamosan beállítja a *fix* logikai mátrix celláinak értékeit is.
- A *matrixBefejez* függvény a nem kitöltött cellákba rak értékeket az $[1, N^2]$ intervallumból.
- A *kovNemSzereplo* függvény megkeresi *k*-tól kiindulva a következő, a mátrixban még nem szereplő számértéket.
- A *Buvos_Kiiras_3* függvény hasonlóan a korábbiakhoz színesen kiírja a mátrix elemeit, valamint a sorok, oszlopok, átlók összegeit. A fix cellákat színes háttérrel, a közönséges cellákat fekete háttérrel jeleníti meg.
- A *valasztElem2* függvény az egymással ütköző *i* vagy *j* sorból választ cellát, ügyelve arra, hogy ne válasszunk fix cellát.

13.9. ábra. A program outputja 1.

13.10. ábra. A program outputja 2.

13.26. forráskód. Anti bűvös négyzet generálás – elem összekeverése

```
static void Main()
{
   int[,] m;
   bool[,] fix;
   beolvasas(out m,out fix,@"hianyos-anti-magikus-negyzet.txt");
   int N = m.GetLength(0);
   int[] osszegek = new int[2 * N + 2];
```

```
   Console.Clear();
   Magikus_Kiiras_3(m, fix);
   Console.SetCursorPosition(0, N + 2);
   Console.WriteLine("Kezdeti allapot beolvasva (nyomj le egy billt)");
   Console.ReadLine();
   //
   matrixBefejez(m);
   Console.Clear();
   Magikus_Kiiras_3(m, fix);
   Console.SetCursorPosition(0, N + 2);
   Console.WriteLine("Kezdeti allapot befejezve (nyomj le egy billt)");
   Console.ReadLine();
   //
   ulong fdb = 0;
   while (true)
   {
      int i, j;
      osszegekSzamol(m, osszegek);
      if (egyformaIndexek(osszegek, out i, out j))
      {
         int a, b, c, d;
         valasztElem2(i, j,out a, out b, N, fix);
         while (true)
         {
            c = rnd.Next(0, N);
            d = rnd.Next(0, N);
            if (fix[c, d] == false) break;
         }
         int x = m[a, b];
         m[a, b] = m[c, d];
         m[c, d] = x;
      }
      else break;
      if (fdb % 100000 == 0)
      {
         Console.Clear();
         Magikus_Kiiras_3(m,fix);
         Console.SetCursorPosition(0, N + 1);
         Console.Write(fdb);
      }
      fdb++;
   }
   //
   Console.Clear();
   Magikus_Kiiras_3(m,fix);
   Console.SetCursorPosition(0, N + 2);
   Console.WriteLine("{0} | K É S Z ! ", fdb);
   Console.ReadLine();
}
```

13.27. forráskód. Anti bűvös négyzet generálás – beolvasás

```
static void beolvasas(out int[,] m, out bool[,] fix, string fnev)
{
   m = null;
   fix = null;
   int N = 0;
   int i = 0;
   System.IO.StreamReader r =
    new System.IO.StreamReader(fnev, System.Text.Encoding.Default);
   while (!r.EndOfStream)
   {
      string s = r.ReadLine().Trim();
      string[] ss = s.Split(' ');
      if (m == null)
      {
         N = ss.Length;
         m = new int[N, N];
         fix = new bool[N, N];
      }
      if (ss.Length != N)
         throw new Exception("File feldolgozasi hiba");
      if (i>=N)
         throw new Exception("Tul sok sor a file-ban");
      for (int j = 0; j < N; j++)
      {
         m[i, j] = int.Parse(ss[j]);
         fix[i, j] = (m[i, j] != 0);
      }
      i++;
   }
   r.Close();
   if (m==null)
      throw new Exception("File ures volt");
}
```

13.28. forráskód. Anti bűvös négyzet generálás – a maradék cellák kitöltése

```
static void matrixBefejez(int[,] m)
{
   int N = m.GetLength(0);
   int k = 1;
   for(int i=0;i<N;i++)
   {
      for (int j = 0; j < N; j++)
      {
         if (m[i, j] == 0)
```

```
            {
               k = kovNemSzereplo(m, k);
               m[i, j] = k;
            }
         }
      }
   }
```

13.29. forráskód. Anti bűvös négyzet generálás – következő még nem szereplő érték keresése

```
static int kovNemSzereplo(int[,] m, int k)
{
   int N = m.GetLength(0);
   while (true)
   {
      int db = 0;
      for (int i = 0; i < N && db==0; i++)
         for (int j = 0; j < N && db == 0; j++)
            if (k == m[i, j]) db++;
      if (db == 0) return k;
      k++;
   }
}
```

13.30. forráskód. Anti bűvös négyzet generálás – megjelenítés

```
static void Magikus_Kiiras_3(int[,] m, bool[,] fix)
{
   int N = m.GetLength(0);
   int[] osszegek = new int[2 * N + 2];
   osszegekSzamol(m, osszegek);
   int x,y;
   egyformaIndexek(osszegek, out x, out y);

   //
   Console.ForegroundColor = ConsoleColor.Yellow;
   for (int i = 0; i < N; i++)
   {
      Console.Write("  ");
      for (int j = 0; j < N; j++)
      {
         if (fix[i, j])
         {
            Console.BackgroundColor = ConsoleColor.Green;
            Console.ForegroundColor = ConsoleColor.Black;
         }
         else
         {
            Console.BackgroundColor = ConsoleColor.Black;
            Console.ForegroundColor = ConsoleColor.Yellow;
         }
         Console.Write("{0,3} ", m[i, j]);
         Console.BackgroundColor = ConsoleColor.Black;
      }
      Console.WriteLine();
   }
   // sorok osszegei
   for(int i=0;i<N;i++)
   {
      if (i==x || i==y) Console.ForegroundColor = ConsoleColor.Red;
      else Console.ForegroundColor = ConsoleColor.Cyan;
      Console.SetCursorPosition(N*4+4,i);
      Console.Write("{0,3} ", osszegek[i]);
   }


   // oszlopok osszegei
   for (int i = 0; i < N; i++)
   {
      if (i+N == x || i+N == y)
         Console.ForegroundColor = ConsoleColor.Red;
      else
         Console.ForegroundColor = ConsoleColor.Cyan;
      Console.SetCursorPosition(4+i*4, N);
      Console.Write("{0,3} ", osszegek[i+N]);
   }
   // foatlo
   if (2*N == x || 2*N == y)
      Console.ForegroundColor = ConsoleColor.Red;
   else
      Console.ForegroundColor = ConsoleColor.Green;
   Console.SetCursorPosition(4 + N * 4, N);
   Console.Write("{0,3} ", osszegek[2*N]);
   // mellektalo
   if (2 * N+1 == x || 2 * N+1 == y)
     Console.ForegroundColor = ConsoleColor.Red;
   else
     Console.ForegroundColor = ConsoleColor.Green;
   Console.SetCursorPosition(0, N);
   Console.Write("{0,3} ", osszegek[2 * N+1]);
   Console.ForegroundColor = ConsoleColor.Gray;
}
```

13.31. forráskód. Anti bűvös négyzet generálás – cserélendő cellák választása

```
static void valasztElem2(int i, int j,out int a, out int b,
                  int N, bool[,] fix)
{
   while (true)
   {
```

```
            valasztElem(i, out a, out b, N);
            if (fix[a, b] == false) return;
            valasztElem(j, out a, out b, N);
            if (fix[a, b] == false) return;
        }
    }
```

Bevezető információk: A *szép* antimágikus négyzetek alatt értsük azt az eset, amikor az egyes sorokban szereplő értékek egymást követő természetes számok, valamint (hasonlóan) az oszlopokban szereplő értékek összegei is egymást követő természetes számok!

13.9. feladat (Szép antimágikus négyzet ellenőrzése – *szint: 2*). Készítsünk olyan függvényt, amely beolvas fájlból egy $N \times N$ méretű mátrixot, majd ellenőrzi, hogy az *szép* antimágikus négyzet-e!

Magyarázat: A 13.68. függvény képes beolvasni mátrixot fájlból. A 13.58. függvény képes eldönteni, hogy antimágikus négyzet-e. A 13.61. forráskódban bemutatott *osszegekSzamol* függvény képes előállítani a sorok és oszlopok összegeit. Az összegek ellenőrzése ettől kezdve már nem nehéz (lásd a 13.74. forráskód).

13.32. forráskód. Szép anti bűvös négyzet ellenőrzés

```
static bool szep_magikus_negyzet_e(int[,] m)
{
   int N = m.GetLength(0);
   int[] osszegek = new int[2 * N + 2];
   osszegekSzamol(m, osszegek);
   // sorok osszegei
   for (int i = 1; i < N; i++)
      if (osszegek[i - 1] != osszegek[i] + 1)
         return false;
   // oszlopok osszegei
   for (int i = N; i < 2*N; i++)
      if (osszegek[i - 1] != osszegek[i] + 1)
         return false;
   // minden rendben
   return true;
}
```

Bevezető információk: A magyar kártyában négy szín (piros, zöld, makk, tök) fordul elő, mindegyik színből van számozott (7...10) és figurás (alsó, felső, király, ász) lap, így összesen 32 lapos. Figurás lapokból 4 különböző van, ha mind a 4 szín figurás lapjait összeszedjük, akkor pontosan 16 lapot kapunk.

13.10. feladat (Kártyaalapú mágikus négyzet – *szint: 3*). Helyezzük el a magyar kártya figurás lapjait egy 4 x 4-es mágikus négyzetben úgy, hogy minden sorban, minden oszlopban, valamint a két átlóban is különböző színű és különböző figurás lapok legyenek!

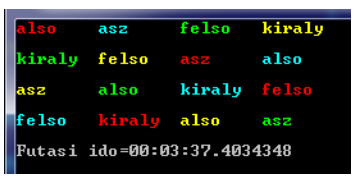
13.11. ábra. Egy lehetséges megoldás

Magyarázat: Természetesen konzolos felületen nem lehet ilyen szép outputot generálni, mint amit a 13.11. ábrán láthatunk. Alkalmazzunk kódolást a kártyákra. Jelöljük 11, 12, 13, 14-gyel a piros színű alsó, felső, király, ász lapokat! Hasonlóan 21, 22, 23, 24 értékekkel a zöld alsó, felső, király, ász, 31, 32, 33, 34-gyel a makk, 41, 42, 43, 44 értékkel a tök színű lapokat. A szabályt ekkor úgy fogalmazhatjuk meg: egyetlen sorban, oszlopban, átlóban sem fordulhat elő két olyan számérték, melyeknek 10-zel való egész vagy maradékos osztásának eredménye egyenlő. Ha a 10-zel való egész osztás eredménye egyenlő lenne, akkor az egyforma színt jelentene. Ha a modulo 10 értéke egyenlő, akkor az egyforma figurát jelentene.

Készítsük el a mátrix egy alapkitöltését, helyezzük el benne az összes számértéket módszeresen sorfolytonosan! Az így feltöltött mátrix nyilván nem felel meg a feltételeknek. Válasszunk ki két cellát véletlenszerűen, és cseréljük fel a bennük lévő értékeket, mindaddig, míg meg nem kapjuk a kért tulajdonságú végeredményt! Ez lényegében egyezik az antimágikus négyzet generálásánál bemutatott módszerrel.

Sajnos gyakorlatilag reménytelen, hogy a véletlen cserék révén helyes mátrixot kapjunk. Ezért hasonlóan, most is meg kell keresnünk, melyik sorok vagy mely oszlopok hiúsítják meg a kívánt végeredmény elérését, azon belül melyik két cella okozza a konfliktust. Elegendőek az egyik problémás cella koordinátái. Válasszunk hozzá véletlenszerűen egy másik cellát a mátrixból, és cseréljük fel e két cella tartalmát! A módszer elvileg működőképes, de gyakorlatilag nem lehet megmondani, mennyi időbe kerül. A futási idő egy dupla magos gépen futtatva körülbelül 3 perc és 5 perc között változott. A program outputja a 13.76. ábrán látható, a 13.75. ... 13.82. forráskódok tartalmazzák a megoldást.

13.12. ábra. Egy lehetséges megoldás



13.33. forráskód. Kártyás – főprogram

```
static void Main()
{
   const int N = 4;
   int[,] m = new int[N,N];
   kezdoFeltoltes(m);
   ulong fdb = 0;
   int v,w;
   DateTime start = DateTime.Now;
   while (matrixRendben(m, out v, out w ) == false)
   {
      int x = rnd.Next(0, N);
      int y = rnd.Next(0, N);
      //
      int c = m[x, y];
      m[x, y] = m[v, w];
      m[v, w] = c;
      //
      // fdb++;
      // if (fdb % 1000000 == 0) Megjelenit(m);
   }
   DateTime stop = DateTime.Now;
   TimeSpan kul = stop - start;
   Megjelenit(m);
   Console.WriteLine();
   Console.WriteLine();
   Console.ForegroundColor = ConsoleColor.Gray;
   Console.WriteLine("Futasi ido={0}", kul);
   Console.ReadLine();
}
```

13.34. forráskód. Kártyás – megjelenítés

```
static void Megjelenit(int[,] m)
{
   Console.Clear();
   int N = m.GetLength(0);
   ConsoleColor[] szinek = new ConsoleColor[]
      { ConsoleColor.Green, ConsoleColor.Red,
        ConsoleColor.Cyan, ConsoleColor.Yellow};
   string[] lapok = new string[] { "also", "felso", "kiraly", "asz" };
   for (int i = 0; i < N; i++)
   {
      for (int j = 0; j < N; j++)
      {
         Console.SetCursorPosition(j * 8, i * 2);
         int x = m[i, j];
         int v = x / 10 - 1;
         int w = x % 10 - 1;
         Console.ForegroundColor = szinek[v];
         Console.Write(lapok[w]);
      }
   }
}
```

13.35. forráskód. Kártyás – a mátrix megfelelőségének ellenőrzése

```
static bool matrixRendben(int[,] m, out int k, out int l)
{
   k = -1;
   l = -1;
   int N = m.GetLength(0);
   for (int i = 0; i < N; i++)
   {
      if (sorRendben(m, i, N, out l) == false)
      {
        k = i;
        return false;
      }
      if (oszlopRendben(m, i, N, out k) == false)
      {
       l = i;
       return false;
      }
   }
   if (atlokRendben(m, N,out k, out l) == false)
      return false;
   return true;
}
```

13.36. forráskód. Kártyás – átlók megfelelőségének ellenőrzése

```
static bool atlokRendben(int[,] m, int N, out int v, out int w)
{
   v = -1;
   w = -1;
   for (int i = 0; i < N; i++)
    for (int j = i + 1; j < N; j++)
    {
      if (hasonlo(m[i, i], m[j, j]))
      {
        v = i;
        w = i;
        return false;
      }
      if (hasonlo(m[i, N - i - 1], m[j, N - j - 1]))
      {
        v = i;
```

```
            w = N - i - 1;
            return false;
         }
      }
      //
      return true;
   }
```

13.37. forráskód. Kártyás – a mátrix *k*. oszlopának ellenőrzése

```
static bool oszlopRendben(int[,] m, int k, int N, out int w)
{
   w = -1;
   for (int i = 0; i < N; i++)
      for (int j = i + 1; j < N; j++)
         if (hasonlo(m[i, k], m[j, k])) { w = i; return false; }
   //
   return true;
}
```

13.38. forráskód. Kártyás – a mátrix *k*. sorának ellenőrzése

```
static bool sorRendben(int[,] m, int k, int N, out int w)
{
   w = -1;
   for (int i = 0; i < N; i++)
      for (int j = i + 1; j < N; j++)
         if (hasonlo(m[k, i], m[k, j])) { w = i; return false; }
   //
   return true;
}
```

13.39. forráskód. Kártyás – két cella színének és lapjának vizsgálata

```
static bool hasonlo(int a, int b)
{
   if (a / 10 == b / 10 || a % 10 == b % 10) return true;
   return false;
}
```

13.40. forráskód. Kártyás – a mátrix kezdő feltöltése

```
static void kezdoFeltoltes(int[,] m)
{
   int N = m.GetLength(0);
   for (int i = 0; i < N; i++)
   {
      int k = (i+1)*10+1;
      for (int j = 0; j < N; j++)
      {
         m[i, j] = k;
         k++;
      }
   }
}
```

Bevezető információk: Egy $N \times N$ mátrixot *latin négyzetnek* nevezünk, ha minden sorában és minden oszlopában – tetszőleges sorrendben – ugyanaz az *N* darab szám áll, de mindegyik csak egyszer.

Tehát nem kell a mágikus négyzetekhez hasonlóan minden mezőbe különböző számot írni, és – általában – az átlókra sincs kikötés.

$$\begin{pmatrix} a & b \\ b & a \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{pmatrix}$$

13.11. feladat (Latin négyzet – *szint: 2*). Generáljunk egy $N \times N$ méretű latin négyzetet, ahol *N* értékét a program indulásakor a felhasználó adja meg! A generált négyzetet táblázatos alakban írjuk ki a *latin-negyzet.txt* fájlba!

Magyarázat: Ez utóbbi könnyítések miatt a latin négyzetek kitöltése rendkívül egyszerű: az első sorba írjuk tetszőleges sorrendben az 1, 2, ..., n számokat, majd lefelé haladva a következő sorba – a sorrendet megtartva – eggyel jobbra (vagy balra) tolva írjuk le, azaz ciklikusan permutáljuk. A megoldás a 13.83. forráskódban olvasható.

Bevezető információk: A matematikában egy mágikus négyzetet *másodfokúnak*[5] nevezhetünk, ha mágikus négyzet marad akkor is, ha a benne szereplő minden *n* számot négyzetre emeljük. Harmadfokúnak[6] nevezhetjük, ha minden számot harmadik hatványára emelve az szintén mágikus négyzet marad. A harmadfokúak nem feltétlenül másodfokúak. Általában véve, *k*-ad fokúnak nevezünk egy mágikus négyzetet, ha minden számot *k*. hatványra emelve szintén egy mágikus négyzetet kapunk.

13.12. feladat (K-ad fokú mágikus négyzetek – *szint: 2*). Készítsünk olyan függvényt, amely egy adott $N \times N$ méretű mágikus négyzetről eldönti, hogy még milyen *k*-ad fokú mágikus négyzet is egyben! A függvény kimenete egy lista, melyben a *k* értékei szerepelnek. Ez a lista legyen üres, ha a mágikus négyzetünk semmilyen *k* értékre nem *k*-ad fokú! A *k* értékeit [1 ... 10] között vizsgálja a program!

Magyarázat: A feladat nagyon egyszerű. A 13.43. kódban megadott ellenőrző függvény képes ellenőrizni, hogy egy *m* mátrix mágikus négyzet-e. Mindössze elő kell állítani a mátrixelemek különböző hatványát, és alkalmazni kell az ellenőrző függvényt (lásd a 13.84. forráskódot).

Bevezető információk: Egy négyzetet *bűvös négyzetnek* nevezünk, ha mágikus négyzet, és az N×N méretű mátrix celláiban az összes $[1...N^2]$ szám szerepel (13.87. ábra).

13.13. ábra. Bűvös négyzet

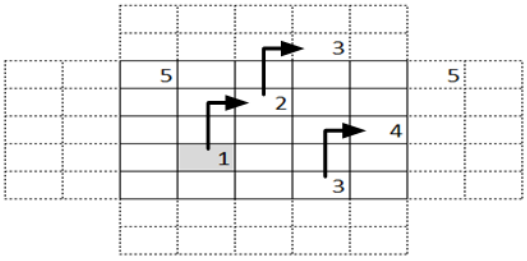
16	2	3	13	34
5	11	10	8	34
9	7	6	12	34
4	14	15	1	34
34	34	34	34	

13.13. feladat (Bűvös négyzet-e – *szint: 2*). Készítsünk programot, amely egy $N \times N$ méretű mátrixot beolvas fájlból, majd eldönti, hogy a mátrix bűvös négyzet-e!

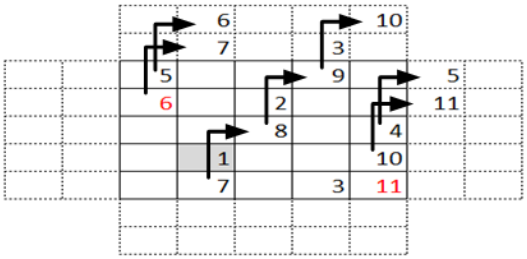
Magyarázat: A 13.43. forráskódban leírt mágikusnégyzet-ellenőrző függvényt kell annyiban bővíteni, hogy $[1, N^2]$ közötti minden szám szerepel-e a mátrixban (pontosan egyszer szerepel-e). Mivel a mátrix mérete is N^2, így ha minden szám szerepel a szóban forgó intervallumból, akkor egyúttal pontosan egyszer szerepel. A plusz ellenőrzés kódja (melyet be lehet szúrni a 13.43. kódban megadott függvény belsejébe) a 13.85. forráskódban olvasható.

Bevezető információk: A misztikus négyzetek előállításának több módszere is van. Általában külön szokták venni a páros és a páratlan N méretű mátrixok előállítási módszereit. Ha N páratlan, akkor a *huszármódszerrel* próbálkozhatunk. Ennek során írjuk be a mátrix tetszőleges pontjára az 1 értéket, majd a sakkban ismert huszár **L** lépési technikájával lépjünk 2-t fel 1-et jobbra, és ide írjuk a következő számot! Ha eközben kilépnénk a mátrixból, akkor úgy kell kezelni a cellát, hogy ott is egy ugyanilyen mátrix áll, és az abba eső pozíciót kell az itteni mátrixban felölteni. Ha az a cella már foglalt, akkor a lóugrás kiindulási pontja alatti cellába kell írni (lásd a 13.14. és a 13.15. ábrák).

13.14. ábra. Huszár módszer 1. lépés



13.15. ábra. Huszár módszer folytatás



13.14. feladat (Huszármódszer – *szint: 4*). Kérjük be *N* értékét billentyűzetről, $N \in [3,21]$, és páratlan. Készítsük el az $N \times N$ méretű misztikus mátrixot a huszármódszerrel, jelenítsük meg táblázatos alakban a képernyőn, és ellenőrizzük le, hogy valóban teljesíti-e a misztikus mátrixok tulajdonságait!

Az algoritmus alapján készült 13.86. forráskód tartalmazza a huszármódszer C# nyelvi kódját. Az utólagos ellenőrzést a 13.85. forráskódban található függvénnyel végezhetjük el. A program futását a 13.1. videón követketjük nyomon.

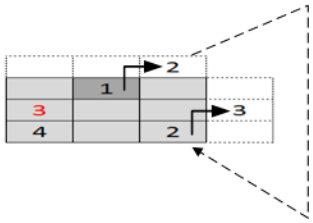
13.1. videó. A huszármódszer futási képernyője

Bevezető információk: Az ún. *lépcsős módszer*[7] nagyon sokban hasonlít a huszármódszerre, de ennek során felfele nem kettőt, csak 1-et kell lépni, és kezdőpontja kötelezően adott. Segítségével ugyanúgy páratlan N méretű bűvös négyzeteket lehet generálni. A módszer az alábbi lépésekből áll:

- Írjuk be az 1 értéket a felső sor középső eleméhez!
- Lépjünk fel és jobbra egy lépéssel, az ottani cellába! Ha közben kilépnénk a mátrixból, akkor ciklikusan balról ⟶ jobbra, fentről ⟶ lefele kötve a cellákat visszaléphetünk a mátrixba.
- Ha üres a cella, akkor oda írjuk be a soron következő számot!
- Ha a cella nem üres, akkor lépjünk az eredeti cellából lefele (ciklikusan ha alul kilépnénk, akkor fenn lépjünk be újra)!
- Ismételjük a lépéseket, amíg az összes cella be nem telik!

Ha mindent jól csináltunk, a legmagasabb szám a mátrix legalsó sorának közepére fog esni (itt kell befejeznünk a generálást).

13.16. ábra. Lépcsős módszer



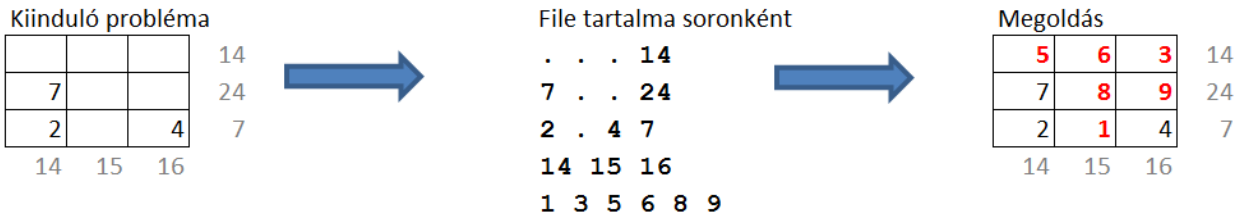
13.15. feladat (Lépcsős generáló módszer – *szint: 2*). Készítsünk olyan programot, amely bekéri *N* értékét, ahol $N \in [3,21]$ és *N* páratlan! Generáljunk bűvös négyzetet a lépcsős módszer segítségével, és jelenítsük meg a képernyőn táblázatos formában!

Magyarázat: Az algoritmus alapján a kód elkészítése nem különösebben nehéz (13.87. forráskód).

Bevezető információk: A mindennapi életben a keresztrejtvények szintjén a bűvös négyzetet sokkal szabadabban értelmezik. Ott gyakran csak egy olyan négyzetre gondolnak, melynek celláiban számok szerepelnek, mindegyik csak egyszer, és a sorok és oszlopok összegei egyeznek egy (az adott sorhoz és oszlophoz külön-külön megadott) értékkel. Tehát korántsem kikötés, hogy ezen oszlop- és sorösszegek egymással egyezzenek, illetve jellemzően nincs szó az átlókban szereplő összegekről. Ezen rejtvényfeladványok esetén a mátrixok kitöltését valahány elemig meg is adják, a feladvány csak ezen kezdemény jó *befejezése*. Ez persze papíron ceruzával ettől még igazi feladvány, melynek megoldása a fejben számolást mindenképpen fejleszti, de egyéb logikai képességeket is, és igazi sikerélményhez juttat.

13.16. feladat (Keresztrejtvény bűvös mátrixa – *szint: 3*). Egy $N \times N$ méretű mátrix kezdő feltöltését egy szöveges fájlban adjuk meg. A fájlban soronként N + 1 cella értéke szerepel, mely kétféle lehet: vagy egy szám szerepel (a cella értéke), vagy a nem kitöltött cellákat egy . (pont) karakter jelöli. Az utolsó szám a sorokban nem valamely cella értéke, hanem az adott sor elvárt sorösszege (és emiatt sosem pont, hanem konkrét szám szerepel ott). A mátrix méretét az első sorában szereplő számok és pont karakterek számából állapíthatjuk meg. A fájlban az N + 1 sorban a mátrix N oszlopának elvárt oszlopösszegei szerepelnek (emiatt itt csak N számérték szerepel, nem N + 1). A fájl további sorában újabb számok vannak szóközzel határolva, melyek mennyisége pontosan egyezik a korábban a mátrix nem kitöltött (pont karakterrel jelölt) celláinak számával. Ezen *felhasználható számok* listája alapján próbáljunk meg a kitöltetlen cellákba számokat írni oly módon, hogy a korábban megadott sor- és oszlopösszegek előálljanak! Minden felhasználható számot pontosan egyszer használhatunk fel valamely cella feltöltésére. A már kitöltött cellákban lévő számokat nem cserélhetjük ki. Ha a mátrix nem megoldható (nincs olyan feltöltés, mellyel a sor- és oszlopösszegek előállnak), akkor jelezzük (a feladat értelmezéséhez segítség a 13.84. ábra)!

13.17. ábra. Keresztrejtvény-feladvány



Magyarázat: A feladat nagyban hasonlít a 13.8. feladatban leírt antimágikus négyzet befejezéséhez. Ott is $[1, N^2]$ közötti kell számokkal kell kiegészíteni a mátrixot, el kell különíteni a fixen rögzített cellákat a szabad celláktól. A különbség mindössze az, mikor kapunk *megfelelő* kitöltést. A megoldással kapcsolatos C# függvények az 13.88... 13.94. forráskódokban olvashatóak. A program futását a 13.2. képernyőn követhetjük nyomon.

13.2. videó. A keresztrejtvény-feladat megoldása

A fejezet forráskódjai

13.41. forráskód. Duplán sztochasztikusság ellenőrzése klasszikus megoldással

```
static bool duplan_sztochasztikus(int[,] m)
{
   int N = m.GetLength(0);
   int M = m.GetLength(1);
   // minden eleme nemnegatív ?
   for (int i = 0; i < N; i++)
      for (int j = 0; j < M; j++)
         if (m[i, j] < 0) return false;
   // sorosszeg mindenütt 1.0 ?
   for (int i = 0; i < N; i++)
   {
      double sum = 0.0;
      for (int j = 0; j < M; j++)
         sum = sum + m[i, j];
      if (sum != 1.0) return false;
   }
   // oszlop osszegek mindenütt 1.0 ?
   for (int j = 0; j < M; j++)
   {
      double sum = 0.0;
      for (int i = 0; i < N; i++)
         sum = sum + m[i, j];
      if (sum != 1.0) return false;
   }
   // minden rendben
   return true;
}
```

13.42. forráskód. Duplán sztochasztikusság ellenőrzése optimalizált módon

```
static bool duplan_sztochasztikus_opt(int[,] m)
{
   int N = m.GetLength(0);
   int M = m.GetLength(1);
```

```
   double[] sorok = new double[N];
   double[] oszlopok = new double[M];
   // matrix bejarasa
   for (int i = 0; i < N; i++)
      for (int j = 0; j < M; j++)
      {
         if (m[i, j] < 0) return false;
         sorok[i] = sorok[i] + m[i, j];
         oszlopok[j] = oszlopok[j] + m[i, j];
      }
   // ellenorzes
   for (int i = 0; i < N; i++)
   if (sorok[i]!=1.0) return false;
   for (int j = 0; j < M; j++)
      if (oszlopok[j] != 1.0) return false;
   // minden rendben
   return true;
}
```

13.43. forráskód. Bűvös négyzet – „buvos_negyzet_e”

```
   static bool magikus_negyzet_e(int[,] m)
   {
      int N = m.GetLength(0);
      int[] sorok = new int[N];
      int[] oszlopok = new int[N];
      osszegekGen(m, sorok, oszlopok);
      // sorok, oszlopok osszegei
      if (mindEgyenlo(sorok) == false ||
         mindEgyenlo(oszlopok) == false ||
         sorok[0] != oszlopok[0]) return false;
      // atlok osszegei
      int atlo1, atlo2;
      atlokGen(m, out atlo1, out atlo2);
      if (atlo1 != atlo2 || atlo1 != sorok[0])
         return false;
      // egyediseg
      for (int i = 0; i < N; i++)
         for (int j = 0; j < N; j++)
            if (megszamol(m, m[i, j]) != 1)
               return false;
      // minden rendben
      return true;
   }
```

13.44. forráskód. Bűvös négyzet – „osszegekGen”

```
static void osszegekGen(int[,] m, int[] sorok, int[] oszlopok)
{
 for (int i = 0; i < sorok.Length; i++)
   for (int j = 0; j < oszlopok.Length; j++)
   {
   sorok[i] = sorok[i] + m[i, j];
   oszlopok[j] = oszlopok[j] + m[i, j];
   }
}
```

13.45. forráskód. Bűvös négyzet – „mindEgyenlo”

```
   static bool mindEgyenlo(int[] l)
   {
      int x = l[0];
      foreach (int a in l)
         if (x != a) return false;
      return true;
   }
```

13.46. forráskód. Bűvös négyzet – „atlokGen”

```
   static void atlokGen(int[,] m, out int a, out int b)
   {
      a = 0; b = 0;
      int N = m.GetLength(0);
      for (int i = 0; i < N; i++)
      {
         a = a + m[i, i];
         b = b + m[i, N - i - 1];
      }
   }
```

13.47. forráskód. Bűvös négyzet – „megszamol”

```
   static int megszamol(int[,] m, int x)
   {
      int N = m.GetLength(0);
      int db = 0;
      for (int i = 0; i < N; i++)
         for (int j = 0; j < N; j++)
            if (m[i, j] == x) db++;
      return db;
   }
```

13.48. forráskód. A mátrix értékeinek ellenőrzése

```
bool mindegyikPrim = true;
```

```
for(int i=0;i<N && mindegyikPrim;i++)
{
 for(int j=0;j<M && mindegyikPrim;j++)
 {
   if (prim-e(m[i,j]==false)
      mindegyikPrim = false;
 }
}
```

13.49. forráskód. Jobbra-le törtátlók

```
   static void panmagikus_kiir_A(int[,] m)
   {
      int N = m.GetLength(0);
      ConsoleColor[] color = new ConsoleColor[] { ConsoleColor.Red,
          ConsoleColor.Green, ConsoleColor.Yellow, ConsoleColor.Cyan,
          ConsoleColor.White, ConsoleColor.Magenta };
      //
      int szin = 0;
      for (int i = 0; i < N; i++)
      {
         szin = i;
         for (int j = 0; j < N; j++)
         {
            Console.ForegroundColor = color[szin];
            Console.Write("{0,4}", m[i, j]);
            szin++;
            if (szin >= N) szin = 0;
         }
         Console.WriteLine();
      }
  }
```

13.50. forráskód. Balra-le törtátlók

```
   static void panmagikus_kiir_A(int[,] m)
   {
      int N = m.GetLength(0);
      ConsoleColor[] color = new ConsoleColor[] { ConsoleColor.Red,
          ConsoleColor.Green, ConsoleColor.Yellow, ConsoleColor.Cyan,
          ConsoleColor.White, ConsoleColor.Magenta };
      //
      int szin = 0;
      for (int i = 0; i < N; i++)
      {
         szin = i;
         for (int j = 0; j < N; j++)
         {
            Console.ForegroundColor = color[szin];
            Console.Write("{0,4}", m[i, j]);
            szin++;
            if (szin >= N) szin = 0;
         }
         Console.WriteLine();
      }
  }
```

13.51. forráskód. Törtátlókban található összegek képzése és összehasonlítása

```
   static bool panmagikus_ell(int[,] m)
   {
      int N = m.GetLength(0);
      int[] sum1 = new int[N];
      int[] sum2 = new int[N];
      //
      int i1, i2;
      for (int i = 0; i < N; i++)
      {
         i1 = i;
         i2 = N - i - 1;
         for (int j = 0; j < N; j++)
         {
            sum1[i1] = sum1[i1] + m[i, j];
            sum2[i2] = sum2[i2] + m[i, j];
            i1++;
            if (i1>= N) i1= 0;
            i2++;
            if (i2 >= N) i2 = 0;
         }
      }
      //
      for (int i = 1; i < N; i++)
         if (sum1[i] != sum1[0])
            return false;
      for (int i = 1; i < N; i++)
         if (sum2[i] != sum2[0])
            return false;
      if (sum1[0] != sum2[0])
         return false;
      // minden ok
      return true;
   }
```

13.52. forráskód. Bűvös négyzet – „buvos_negyzet_e_2”

```
static bool magikus_negyzet_e_2(int[,] m,int eltolas)
{
   int N = m.GetLength(0) - 2 * eltolas;
   int[] sorok = new int[N];
   int[] oszlopok = new int[N];
   osszegekGen2(m, sorok, oszlopok, eltolas);
   // sorok, oszlopok osszegei
   if (mindEgyenlo(sorok) == false ||
      mindEgyenlo(oszlopok) == false ||
      sorok[0] != oszlopok[0]) return false;
   // atlok osszegei
   int atlo1, atlo2;
   atlokGen2(m, out atlo1, out atlo2,eltolas);
   if (atlo1 != atlo2 || atlo1 != sorok[0])
      return false;
   // egyediseg
   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
         if (megszamol2(m, m[i, j],eltolas) != 1)
            return false;
   // minden rendben
   return true;
}
```

13.53. forráskód. Bűvös négyzet – „osszegekGen2”

```
static void osszegekGen2(int[,] m,int[] sorok,int[] oszlopok,int eltolas)
{
  for (int i = 0; i < sorok.Length; i++)
     for (int j = 0; j < oszlopok.Length; j++)
     {
         sorok[i] = sorok[i] + m[i + eltolas, j + eltolas];
         oszlopok[j] = oszlopok[j] + m[i + eltolas, j + eltolas];
     }
}
```

13.54. forráskód. Bűvös négyzet – „mindEgyenlo”

```
static bool mindEgyenlo(int[] l)
{
   int x = l[0];
   foreach (int a in l)
      if (x != a) return false;
   return true;
}
```

13.55. forráskód. Bűvös négyzet – „atlokGen2”

```
static void atlokGen2(int[,] m, out int a, out int b, int eltolas)
{
   a = 0; b = 0;
   int N = m.GetLength(0);
   for (int i = 0; i < N; i++)
   {
      a = a + m[i + eltolas, i + eltolas];
      b = b + m[i + eltolas, N - i - 1 - +eltolas];
   }
}
```

13.56. forráskód. Bűvös négyzet – „megszamol2”

```
static int megszamol2(int[,] m, int x, int eltolas)
{
   int N = m.GetLength(0)-eltolas;
   int db = 0;
   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
        if (m[i+eltolas, j+eltolas] == x) db++;
   return db;
}
```

13.57. forráskód. Antimágikus négyzet-e – teszt főprogram

```
static void Main()
{
   const int N = 5;
   int[,] m = new int[N, N];
   // kezdo feltoltes
   int a = 1;
   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
      {
         m[i, j] = a;
         a++;
      }
   //
   Magikus_Kiiras_2(m);
   //
   Console.SetCursorPosition(0, N + 2);
   if (anti_magikus_negyzet_e(m) == false)
   {
      Console.ForegroundColor = ConsoleColor.Red;
      Console.WriteLine("NEM ANTI-MAGIKUS NEGYZET");
   }
   else
   {
      Console.ForegroundColor = ConsoleColor.Green;
```

```
      Console.WriteLine("IGENIS ANTI-MAGIKUS NEGYZET");
   }
   Console.ReadLine();
}
```

13.58. forráskód. Antimágikus négyzet-e – az ellenőrző függvény

```
static bool anti_magikus_negyzet_e(int[,] m)
{
 int N = m.GetLength(0);
 int[] osszegek = new int[2*N+2];
 // az osszegek kepzese
 for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
   {
    // i. sor osszege
    osszegek[i]  = osszegek[i] + m[i, j];
    // j. oszlop osszege
    osszegek[N+j] = osszegek[i] + m[i, j];
    // foatlo osszege
    if (i==j) osszegek[2 * N] = osszegek[2*N] + m[i, j];
    // mellekatlo osszege
    if (i == N-j-1) osszegek[2 * N+1] = osszegek[2 * N+1] + m[i, j];
   }

 // van-e ket egyforma osszeg ?
 for (int i = 0; i < osszegek.Length; i++)
   for (int j = i + 1; j < osszegek.Length; j++)
    if (osszegek[i] == osszegek[j])
      return false;
 // minden szam szerepel?
 for (int k = 1; k <= N * N; k++)
 {
   int db = 0;
   for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
      if (m[i, j] == k) db++;
   if (db != 1) return false;
 }
 // minden ok
 return true;
}
```

13.59. forráskód. Antimágikus négyzet-e – mátrixmegjelenítés

```
static void Magikus_Kiiras(int[,] m)
{
   int N = m.GetLength(0);
   int[] oszlopok = new int[N];
   int foatlo = 0;
   int mellekatlo = 0;
   //
   //
   for (int i = 0; i < N; i++)
   {
      Console.ForegroundColor = ConsoleColor.Yellow;
      int sum = 0;
      Console.Write("  ");
      for (int j = 0; j < N; j++)
      {
         Console.Write("{0,3} ", m[i, j]);
         //
         sum = sum + m[i, j];
         oszlopok[j] = oszlopok[j] +m[i, j];
         if (i==j) foatlo=foatlo+m[i,j];
         if (i == N - j - 1)
            mellekatlo = mellekatlo + m[i, j];
      }
      //
      Console.ForegroundColor = ConsoleColor.Cyan;
      Console.Write("{0,4} ", sum);
      Console.WriteLine();
   }
   //
   Console.ForegroundColor = ConsoleColor.Green;
   Console.Write("{0,3} ", mellekatlo);
   Console.ForegroundColor = ConsoleColor.Cyan;
   for (int j = 0; j < N; j++)
      Console.Write("{0,3} ", oszlopok[j]);
   Console.ForegroundColor = ConsoleColor.Green;
   Console.Write("{0,3} ", foatlo);
   Console.ForegroundColor = ConsoleColor.Gray;
}
```

13.60. forráskód. Antimágikus négyzet generálása – főprogram

```
static void Main()
{
   const int N = 8;
   int[,] m = new int[N, N];
   int[] osszegek = new int[2 * N + 2];
   kezdoFeltoltes(m);
   osszekeveres(m, N * N * 40);
   //
   ulong fdb = 0;
   while (true)
```

```
   {
      int i, j;
      osszegekSzamol(m, osszegek);
      if (egyformaIndexek(osszegek, out i, out j))
      {
         int a, b, c, d;
         valasztElem(i, out a, out b, N);
         c = rnd.Next(0, N);
         d = rnd.Next(0, N);
         // csere
         int x = m[a, b];
         m[a, b] = m[c, d];
         m[c, d] = x;
      }
      else break;
      // folyamat kozbeni kijelzes
      if (fdb % 100000 == 0)
      {
         Console.Clear();
         Magikus_Kiiras_2(m);
         Console.SetCursorPosition(0, N + 1);
         Console.Write(fdb);
      }
      fdb++;
   }
   //
   Console.Clear();
   Magikus_Kiiras_2(m);
   Console.SetCursorPosition(0, N + 2);
   Console.WriteLine("{0} | K É S Z ! ", fdb);
   Console.ReadLine();
}
```

13.61. forráskód. Antimágikus négyzet generálása – az összegek számítása

```
static void osszegekSzamol(int[,] m, int[] osszegek)
{
   int N = m.GetLength(0);
   for (int i = 0; i < osszegek.Length; i++)
      osszegek[i] = 0;
   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
      {
         // i. sor osszege
         osszegek[i] = osszegek[i] + m[i, j];
         // j. oszlop osszege
         osszegek[N + j] = osszegek[i] + m[i, j];
         // foatlo osszege
         if (i == j)
            osszegek[2 * N] = osszegek[2 * N] + m[i, j];
         // mellekatlo osszege
         if (i == N - j - 1)
            osszegek[2 * N + 1] = osszegek[2 * N + 1] + m[i, j];
      }
}
```

13.62. forráskód. Antimágikus négyzet generálása – mátrix megjelenítése v2

```
static void Magikus_Kiiras_2(int[,] m)
{
   int N = m.GetLength(0);
   int[] osszegek = new int[2 * N + 2];
   osszegekSzamol(m, osszegek);
   int x,y;
   egyformaIndexek(osszegek, out x, out y);

   //
   Console.ForegroundColor = ConsoleColor.Yellow;
   for (int i = 0; i < N; i++)
   {
      Console.Write("  ");
      for (int j = 0; j < N; j++)
      {
         Console.ForegroundColor = ConsoleColor.Yellow;
         Console.Write("{0,3} ", m[i, j]);
      }
      Console.WriteLine();
   }
   // sorok osszegei
   for(int i=0;i<N;i++)
   {
      if (i==x || i==y) Console.ForegroundColor = ConsoleColor.Red;
      else Console.ForegroundColor = ConsoleColor.Cyan;
      Console.SetCursorPosition(N*4+4,i);
      Console.Write("{0,3} ", osszegek[i]);
   }
   // oszlopok osszegei
   for (int i = 0; i < N; i++)
   {
      if (i+N == x || i+N == y)
        Console.ForegroundColor = ConsoleColor.Red;
      else
        Console.ForegroundColor = ConsoleColor.Cyan;
      Console.SetCursorPosition(4+i*4, N);
      Console.Write("{0,3} ", osszegek[i+N]);
   }
   // foatlo
   if (2*N == x || 2*N == y) Console.ForegroundColor = ConsoleColor.Red;
   else Console.ForegroundColor = ConsoleColor.Green;
   Console.SetCursorPosition(4 + N * 4, N);
   Console.Write("{0,3} ", osszegek[2*N]);
   // mellektalo
```

```
   if (2 * N+1 == x || 2 * N+1 == y)
      Console.ForegroundColor = ConsoleColor.Red;
   else
      Console.ForegroundColor = ConsoleColor.Green;
   Console.SetCursorPosition(0, N);
   Console.Write("{0,3} ", osszegek[2 * N+1]);
}
```

13.63. forráskód. Antimágikus négyzet generálása – elem választása

```
static void valasztElem(int i, out int a, out int b, int N)
{
   // i egy sor indexe
   if (i < N)
   {
      a = i;
      b = rnd.Next(0, N);
      return;
   }
   // i egy oszlop indexe
   if (i < 2*N)
   {
      a = rnd.Next(0, N);
      b = i-N;
      return;
   }
   // i a foatlo
   if (i == 2 * N)
   {
      a = rnd.Next(0, N);
      b = a;
      return;
   }
   // i a mellekatlo
   a = rnd.Next(0, N);
   b = N - a - 1;
}
```

13.64. forráskód. Antimágikus négyzet generálása – mátrix kezdőfeltöltése

```
static void kezdoFeltoltes(int[,] m)
{
   int N = m.GetLength(0);
   int k = 1;
   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
      {
         m[i, j] = k;
         k++;
      }
}
```

13.65. forráskód. Antimágikus négyzet generálása – elem összekeverése

```
static void osszekeveres(int[,] m, int db)
{
   int N = m.GetLength(0);
   while (db > 0)
   {
      int x = rnd.Next(0, N);
      int y = rnd.Next(0, N);
      int v = rnd.Next(0, N);
      int w = rnd.Next(0, N);
      //
      int c = m[x, y];
      m[x, y] = m[v, w];
      m[v, w] = c;
      //
      db--;
   }
}
```

13.66. forráskód. Antimágikus négyzet generálása – egyforma index-e

```
static bool egyformaIndexek(int[] v, out int a, out int b)
{
   a = -1;
   b = -1;
   for (int i = 0; i < v.Length; i++)
      for (int j = i + 1; j < v.Length; j++)
         if (v[i] == v[j])
         {
            a = i;
            b = j;
            return true;
         }
   //
   return false;
}
```

13.67. forráskód. Antimágikus négyzet generálása – elem összekeverése

```
static void Main()
{
   int[,] m;
   bool[,] fix;
   beolvasas(out m,out fix,@"hianyos-anti-magikus-negyzet.txt");
```

```
   int N = m.GetLength(0);
   int[] osszegek = new int[2 * N + 2];
   Console.Clear();
   Magikus_Kiiras_3(m, fix);
   Console.SetCursorPosition(0, N + 2);
   Console.WriteLine("Kezdeti allapot beolvasva (nyomj le egy billt)");
   Console.ReadLine();
   //
   matrixBefejez(m);
   Console.Clear();
   Magikus_Kiiras_3(m, fix);
   Console.SetCursorPosition(0, N + 2);
   Console.WriteLine("Kezdeti allapot befejezve (nyomj le egy billt)");
   Console.ReadLine();
   //
   ulong fdb = 0;
   while (true)
   {
      int i, j;
      osszegekSzamol(m, osszegek);
      if (egyformaIndexek(osszegek, out i, out j))
      {
         int a, b, c, d;
         valasztElem2(i, j,out a, out b, N, fix);
         while (true)
         {
            c = rnd.Next(0, N);
            d = rnd.Next(0, N);
            if (fix[c, d] == false) break;
         }
         int x = m[a, b];
         m[a, b] = m[c, d];
         m[c, d] = x;
      }
      else break;
      if (fdb % 100000 == 0)
      {
         Console.Clear();
         Magikus_Kiiras_3(m,fix);
         Console.SetCursorPosition(0, N + 1);
         Console.Write(fdb);
      }
      fdb++;
   }
   //
   Console.Clear();
   Magikus_Kiiras_3(m,fix);
   Console.SetCursorPosition(0, N + 2);
   Console.WriteLine("{0} | K É S Z ! ", fdb);
   Console.ReadLine();
}
```

13.68. forráskód. Antimágikus négyzet generálása – beolvasás

```
static void beolvasas(out int[,] m, out bool[,] fix, string fnev)
{
   m = null;
   fix = null;
   int N = 0;
   int i = 0;
   System.IO.StreamReader r =
    new System.IO.StreamReader(fnev, System.Text.Encoding.Default);
   while (!r.EndOfStream)
   {
      string s = r.ReadLine().Trim();
      string[] ss = s.Split(' ');
      if (m == null)
      {
         N = ss.Length;
         m = new int[N, N];
         fix = new bool[N, N];
      }
      if (ss.Length != N)
         throw new Exception("File feldolgozasi hiba");
      if (i>=N)
         throw new Exception("Tul sok sor a file-ban");
      for (int j = 0; j < N; j++)
      {
         m[i, j] = int.Parse(ss[j]);
         fix[i, j] = (m[i, j] != 0);
      }
      i++;
   }
   r.Close();
   if (m==null)
      throw new Exception("File ures volt");
}
```

13.69. forráskód. Antimágikus négyzet generálása – a maradék cellák kitöltése

```
static void matrixBefejez(int[,] m)
{
   int N = m.GetLength(0);
   int k = 1;
   for(int i=0;i<N;i++)
   {
      for (int j = 0; j < N; j++)
      {
```

```
         if (m[i, j] == 0)
         {
            k = kovNemSzereplo(m, k);
            m[i, j] = k;
         }
      }
   }
}
```

13.70. forráskód. Antimágikus négyzet generálása – következő még nem szereplő érték keresése

```
static int kovNemSzereplo(int[,] m, int k)
{
   int N = m.GetLength(0);
   while (true)
   {
      int db = 0;
      for (int i = 0; i < N && db==0; i++)
         for (int j = 0; j < N && db == 0; j++)
            if (k == m[i, j]) db++;
      if (db == 0) return k;
      k++;
   }
}
```

13.71. forráskód. Antimágikus négyzet generálása – megjelenítés (1. rész)

```
static void Magikus_Kiiras_3(int[,] m, bool[,] fix)
{
   int N = m.GetLength(0);
   int[] osszegek = new int[2 * N + 2];
   osszegekSzamol(m, osszegek);
   int x,y;
   egyformaIndexek(osszegek, out x, out y);

   //
   Console.ForegroundColor = ConsoleColor.Yellow;
   for (int i = 0; i < N; i++)
   {
      Console.Write("  ");
      for (int j = 0; j < N; j++)
      {
         if (fix[i, j])
         {
            Console.BackgroundColor = ConsoleColor.Green;
            Console.ForegroundColor = ConsoleColor.Black;
         }
         else
         {
            Console.BackgroundColor = ConsoleColor.Black;
            Console.ForegroundColor = ConsoleColor.Yellow;
         }
         Console.Write("{0,3} ", m[i, j]);
         Console.BackgroundColor = ConsoleColor.Black;
      }
      Console.WriteLine();
   }
   // sorok osszegei
   for(int i=0;i<N;i++)
   {
      if (i==x || i==y) Console.ForegroundColor = ConsoleColor.Red;
      else Console.ForegroundColor = ConsoleColor.Cyan;
      Console.SetCursorPosition(N*4+4,i);
      Console.Write("{0,3} ", osszegek[i]);
   }
/* --- folytatása a következő forráskódban --- */
```

13.72. forráskód. Antimágikus négyzet generálása – megjelenítés (befejezés)

```
/* ---- folytatása a "Magikus_Kiiras_3" függvény kódjának
  static void Magikus_Kiiras_3(int[,] m, bool[,] fix)
  ------------------------------------------------------- */

   // oszlopok osszegei
   for (int i = 0; i < N; i++)
   {
      if (i+N == x || i+N == y)
         Console.ForegroundColor = ConsoleColor.Red;
      else
         Console.ForegroundColor = ConsoleColor.Cyan;
      Console.SetCursorPosition(4+i*4, N);
      Console.Write("{0,3} ", osszegek[i+N]);
   }
   // foatlo
   if (2*N == x || 2*N == y)
      Console.ForegroundColor = ConsoleColor.Red;
   else
      Console.ForegroundColor = ConsoleColor.Green;
   Console.SetCursorPosition(4 + N * 4, N);
   Console.Write("{0,3} ", osszegek[2*N]);
   // mellektalo
   if (2 * N+1 == x || 2 * N+1 == y)
     Console.ForegroundColor = ConsoleColor.Red;
   else
     Console.ForegroundColor = ConsoleColor.Green;
   Console.SetCursorPosition(0, N);
   Console.Write("{0,3} ", osszegek[2 * N+1]);
   Console.ForegroundColor = ConsoleColor.Gray;
}
```

13.73. forráskód. Antimágikus négyzet generálása – cserélendő cellák választása

```
static void valasztElem2(int i, int j,out int a, out int b,
                  int N, bool[,] fix)
{
   while (true)
   {
      valasztElem(i, out a, out b, N);
      if (fix[a, b] == false) return;
      valasztElem(j, out a, out b, N);
      if (fix[a, b] == false) return;
   }
}
```

13.74. forráskód. Szép anti bűvös négyzet ellenőrzés

```
static bool szep_magikus_negyzet_e(int[,] m)
{
   int N = m.GetLength(0);
   int[] osszegek = new int[2 * N + 2];
   osszegekSzamol(m, osszegek);
   // sorok osszegei
   for (int i = 1; i < N; i++)
      if (osszegek[i - 1] != osszegek[i] + 1)
         return false;
   // oszlopok osszegei
   for (int i = N; i < 2*N; i++)
      if (osszegek[i - 1] != osszegek[i] + 1)
         return false;
   // minden rendben
   return true;
}
```

13.75. forráskód. Kártyás – főprogram

```
static void Main()
{
   const int N = 4;
   int[,] m = new int[N,N];
   kezdoFeltoltes(m);
   ulong fdb = 0;
   int v,w;
   DateTime start = DateTime.Now;
   while (matrixRendben(m, out v, out w ) == false)
   {
      int x = rnd.Next(0, N);
      int y = rnd.Next(0, N);
      //
      int c = m[x, y];
      m[x, y] = m[v, w];
      m[v, w] = c;
      //
      // fdb++;
      // if (fdb % 1000000 == 0) Megjelenit(m);
   }
   DateTime stop = DateTime.Now;
   TimeSpan kul = stop - start;
   Megjelenit(m);
   Console.WriteLine();
   Console.WriteLine();
   Console.ForegroundColor = ConsoleColor.Gray;
   Console.WriteLine("Futasi ido={0}", kul);
   Console.ReadLine();
}
```

13.76. forráskód. Kártyás – megjelenítés

```
static void Megjelenit(int[,] m)
{
   Console.Clear();
   int N = m.GetLength(0);
   ConsoleColor[] szinek = new ConsoleColor[]
      { ConsoleColor.Green, ConsoleColor.Red,
        ConsoleColor.Cyan, ConsoleColor.Yellow};
   string[] lapok = new string[] { "also", "felso", "kiraly", "asz" };
   for (int i = 0; i < N; i++)
   {
      for (int j = 0; j < N; j++)
      {
         Console.SetCursorPosition(j * 8, i * 2);
         int x = m[i, j];
         int v = x / 10 - 1;
         int w = x % 10 - 1;
         Console.ForegroundColor = szinek[v];
         Console.Write(lapok[w]);
      }
   }
}
```

13.77. forráskód. Kártyás – a mátrix megfelelőségének ellenőrzése

```
static bool matrixRendben(int[,] m, out int k, out int l)
{
   k = -1;
```

```
   l = -1;
   int N = m.GetLength(0);
   for (int i = 0; i < N; i++)
   {
      if (sorRendben(m, i, N, out l) == false)
      {
        k = i;
        return false;
      }
      if (oszlopRendben(m, i, N, out k) == false)
      {
       l = i;
       return false;
      }
   }
   if (atlokRendben(m, N,out k, out l) == false)
      return false;
   return true;
}
```

13.78. forráskód. Kártyás – átlók megfelelőségének ellenőrzése

```
static bool atlokRendben(int[,] m, int N, out int v, out int w)
{
   v = -1;
   w = -1;
   for (int i = 0; i < N; i++)
    for (int j = i + 1; j < N; j++)
    {
      if (hasonlo(m[i, i], m[j, j]))
      {
        v = i;
        w = i;
        return false;
      }
      if (hasonlo(m[i, N - i - 1], m[j, N - j - 1]))
      {
        v = i;
        w = N - i - 1;
        return false;
      }
    }
   //
   return true;
}
```

13.79. forráskód. Kártyás – a mátrix *k*. oszlopának ellenőrzése

```
static bool oszlopRendben(int[,] m, int k, int N, out int w)
{
   w = -1;
   for (int i = 0; i < N; i++)
      for (int j = i + 1; j < N; j++)
         if (hasonlo(m[i, k], m[j, k])) { w = i; return false; }
   //
   return true;
}
```

13.80. forráskód. Kártyás – a mátrix *k*. sorának ellenőrzése

```
static bool sorRendben(int[,] m, int k, int N, out int w)
{
   w = -1;
   for (int i = 0; i < N; i++)
      for (int j = i + 1; j < N; j++)
         if (hasonlo(m[k, i], m[k, j])) { w = i; return false; }
   //
   return true;
}
```

13.81. forráskód. Kártyás – két cella színének és lapjának vizsgálata

```
static bool hasonlo(int a, int b)
{
   if (a / 10 == b / 10 || a % 10 == b % 10) return true;
   return false;
}
```

13.82. forráskód. Kártyás – a mátrix kezdő feltöltése

```
static void kezdoFeltoltes(int[,] m)
{
   int N = m.GetLength(0);
   for (int i = 0; i < N; i++)
   {
      int k = (i+1)*10+1;
      for (int j = 0; j < N; j++)
      {
         m[i, j] = k;
         k++;
      }
   }
}
```

13.83. forráskód. Latin négyzet feltöltése

```
static void latinFeltoltes(int[,] m)
{
   int N = m.GetLength(0);
   for (int i = 0; i < N; i++)
   {
      int k = 1+ i;
      for (int j = 0; j < N; j++)
      {
         m[i, j] = k;
         k++;
         if (k > N) k = 1;
      }
   }
}
```

13.84. forráskód. K-ad fokú bűvös négyzetek

```
static List<int> K_ad_foku(int[,] m)
{
   List<int> ret = new List<int>();
   int N = m.GetLength(0);
   int[,] mm = (int[,])(m.Clone());
   for (int k = 2; k <= 10; k++)
   {
      // hatvanyozas
      for (int i = 0; i < N; i++)
         for (int j = 0; j < N; j++)
            mm[i, j] = mm[i, j] * m[i, j];
      // ellenorzes
      if (magikus_negyzet_e(mm))
         ret.Add(k);
   }
   return ret;
}
```

13.85. forráskód. Kiegészítő ellenőrzés a bűvös négyzethez

```
for(int k=0;k<N*N;k++)
if (megszamol(m, k) != 1)
  return false;
```

13.86. forráskód. Huszármódszer algoritmusa

```
static void huszarModszer(int[,] m)
{
   int N = m.GetLength(0); // paratlan!
   int y = rnd.Next(0,N);
   int x = rnd.Next(0, N);
   // kezdo pozicio
   m[x, y] = 1;
   //
   int db=1;
   int k = 2;
   while (db < N * N)
   {
      // fel fel jobbra
      int yy = y-2;
      int xx = x+1;
      // kilepunk fenn?
      if (yy < 0) yy = N + yy;
      // kilepunk jobbra
      if (xx>=N) xx=0;
      // ez a cella mar foglalt?
      if (m[xx, yy] != 0)
      {
         xx = x;
         yy = y + 1;
         if (yy >= N) yy = 0;
      }
      if (m[xx, yy] != 0)
         throw new Exception("valami nem stimmel");
      // szam elhelyezese
      m[xx, yy] = k;
      // ez az uj aktualis cella
      x = xx;
      y = yy;
      // szamlalok
      k++;
      db++;
   }
}
```

13.87. forráskód. Lépcsős módszer forráskódja

```
static void lepcsosModszer(int[,] m)
{
   int N = m.GetLength(0); // paratlan!
   int y = 0;
   int x = N / 2;
   // elso sor kozepe
   m[x, y] = 1;
   //
   int db=1;
   int k = 2;
```

```
   while (db < N * N)
   {
      // fel jobbra
      int yy = y-1;
      int xx = x+1;
      // kilepunk fenn?
      if (yy < 0) yy = N - 1;
      // kilepunk jobbra
      if (xx>=N) xx=0;
      // ez a cella mar foglalt?
      if (m[xx, yy] != 0)
      {
         xx = x;
         yy = y + 1;
         if (yy >= N) yy = 0;
      }
      if (m[xx, yy] != 0)
         throw new Exception("valami nem stimmel");
      // szam elhelyezese
      m[xx, yy] = k;
      // ez az uj aktualis cella
      x = xx;
      y = yy;
      // szamlalok
      k++;
      db++;
   }
}
```

13.88. forráskód. Keresztrejtvény – főprogram

```
static void Main()
{
   int[,] m;
   bool[,] fix;
   int[] osszegek;
   List<int> l = new List<int>();
   beolvasas_rejtv(out m, out fix, out osszegek, l, "negyzet.txt");
   int N = m.GetLength(0);
   Magikus_Kiiras_4(m, fix,osszegek);
   Console.SetCursorPosition(0, N + 2);
   Console.WriteLine("Kezdeti allapot beolvasva");
   Console.ReadLine();
   //
   matrixBefejez_4(m,l);
   Console.Clear();
   Magikus_Kiiras_4(m, fix, osszegek);
   Console.SetCursorPosition(0, N + 2);
   Console.WriteLine("Kezdeti allapot befejezve");
   Console.ReadLine();
   //
   ulong fdb = 0;
   int[] sumst = new int[N * 2];
   while (true)
   {
      int i;
      osszegekSzamol_4(m, sumst);
      if (elteroOsszeg(osszegek, sumst, out i))
      {
         int a, b, c, d;
         valasztElem_4(i, out a, out b, N, fix);
         while (true)
         {
            c = rnd.Next(0, N);
            d = rnd.Next(0, N);
            if (fix[c, d] == false) break;
         }
         // csere
         int x = m[a, b];
         m[a, b] = m[c, d];
         m[c, d] = x;
      }
      else break;
      if (fdb % 100000 == 0)
      {
         Console.Clear();
         Magikus_Kiiras_4(m, fix, osszegek);
         Console.SetCursorPosition(50, N + 2);
         Console.Write(fdb);
      }
      fdb++;
   }
   //
   Console.Clear();
   Magikus_Kiiras_4(m, fix, osszegek);
   Console.ReadLine();
}
```

13.89. forráskód. Keresztrejtvény – rejtvény beolvasása

```
static void beolvasas_rejtv(out int[,] m, out bool[,] fix,
         out int[] osszegek, List<int> szamok, string fnev)
{
   m = null;
   fix = null;
   osszegek = null;
   int N = 0;
   int i = 0;
   System.IO.StreamReader r =
      new System.IO.StreamReader(fnev, System.Text.Encoding.Default);
   while (!r.EndOfStream)
   {
```

```
      string s = r.ReadLine().Trim();
      string[] ss = s.Split(' ');
      if (m == null)
      {
         N = ss.Length-1;
         m = new int[N, N];
         fix = new bool[N, N];
         osszegek = new int[2 * N];
      }
      if (i == N) // utolso sor
      {
         for (int j = 0; j < N; j++)
            osszegek[N + j] = int.Parse(ss[j]);
      }
      else if (i == N+1) // felhasznalhato szamok
      {
         for (int j = 0; j < ss.Length; j++)
            szamok.Add(int.Parse(ss[j]));
      }
      else
      {
         if (i > N+1)
            throw new Exception("Tul sok sor a file-ban");
         for (int j = 0; j < N + 1; j++)
         {
            if (j == N) osszegek[i] = int.Parse(ss[j]);
            else
            {
               if (ss[j] == ".") m[i, j] = 0;
               else m[i, j] = int.Parse(ss[j]);
               fix[i, j] = (m[i, j] != 0);
            }
         }
      }
      i++;
   }
   r.Close();
   if (m == null)
      throw new Exception("File ures volt");
}
```

13.90. forráskód. Keresztrejtvény – eltérő összegek keresése

```
static bool elteroOsszeg(int[] osszegek, int[] sumst, out int i)
{
   i = -1;
   for (int k = 0; k < osszegek.Length; k++)
   {
      if (osszegek[k] != sumst[k])
      {
         i = k;
         return true;
      }
   }
   return false;
}
```

13.91. forráskód. Keresztrejtvény – sor- és oszlopösszegek számítása

```
static void osszegekSzamol_4(int[,] m, int[] osszegek)
{
   int N = m.GetLength(0);
   for (int i = 0; i < osszegek.Length; i++)
      osszegek[i] = 0;
   for (int i = 0; i < N; i++)
      for (int j = 0; j < N; j++)
      {
         // i. sor osszege
         osszegek[i] = osszegek[i] + m[i, j];
         // j. oszlop osszege
         osszegek[N + j] = osszegek[N+j] + m[i, j];
      }
}
```

13.92. forráskód. Keresztrejtvény – megjelenítés

```
static void Magikus_Kiiras_4(int[,] m, bool[,] fix, int[] sums)
{
   int N = m.GetLength(0);
   int[] sum2 = new int[2 * N + 2];
   osszegekSzamol_4(m, sum2);
   //
   Console.ForegroundColor = ConsoleColor.Yellow;
   for (int i = 0; i < N; i++)
   {
      Console.Write("  ");
      for (int j = 0; j < N; j++)
      {
         if (fix[i, j])
         {
            Console.BackgroundColor = ConsoleColor.Green;
            Console.ForegroundColor = ConsoleColor.Black;
         }
         else
         {
            Console.BackgroundColor = ConsoleColor.Black;
            Console.ForegroundColor = ConsoleColor.Yellow;
         }
         Console.Write("{0,3} ", m[i, j]);
```

```
            Console.BackgroundColor = ConsoleColor.Black;
         }
         Console.WriteLine();
      }
      // sorok osszegei
      for (int i = 0; i < N; i++)
      {
         if (sum2[i]!=sums[i]) Console.ForegroundColor = ConsoleColor.Red;
         else Console.ForegroundColor = ConsoleColor.Cyan;
         Console.SetCursorPosition(N * 4 + 4, i);
         Console.Write("{0,3} ", sum2[i]);
         Console.ForegroundColor = ConsoleColor.Gray;
         Console.Write("{0,3} ", sums[i]);
      }
      // oszlopok osszegei
      for (int i = 0; i < N; i++)
      {
         if (sum2[N+i] != sums[N+i])
            Console.ForegroundColor = ConsoleColor.Red;
         else
            Console.ForegroundColor = ConsoleColor.Cyan;
         Console.SetCursorPosition(4 + i * 4, N);
         Console.Write("{0,3} ", sum2[i + N]);
         Console.SetCursorPosition(4 + i * 4, N+1);
         Console.ForegroundColor = ConsoleColor.Gray;
         Console.Write("{0,3} ", sums[N+i]);
      }
      Console.ForegroundColor = ConsoleColor.Gray;
   }
```

13.93. forráskód. Keresztrejtvény – kezdő kitöltés befejezése

```
static void matrixBefejez_4(int[,] m, List<int> szamok)
{
   int N = m.GetLength(0);
   int k = 0;
   for (int i = 0; i < N; i++)
   {
      for (int j = 0; j < N; j++)
      {
         if (m[i, j] == 0)
         {
            m[i, j] = szamok[k];
            k++;
         }
      }
   }
}
```

13.94. forráskód. Keresztrejtvény – cserélendő elem választása

```
static void valasztElem_4(int i, out int a, out int b,
                     int N, bool[,] fix)
{
   while (true)
   {
      valasztElem(i, out a, out b, N);
      if (fix[a, b] == false) return;
   }
}
```

---

[1] Nincs igazán konszenzus az angol és magyar elnevezésekben. Angolul létező fogalmak a „magic square” és „mystic square”, ami két hasonló, bár egy ponton különböző dolog. A „mystic square”-ben az is követelmény, hogy az $N \times N$ négyzetben csak $[1, N^2]$közötti számok fordulhatnak elő. Ennek megfelelője jegyzetünkben a „bűvös négyzet”, a „magic square”-t ellenben „mágikus négyzetnek” fordítjuk.

[2] egyben bűvös négyzet

[3] egyben bűvös négyzet

[4] angolul néha *Border Square*-nek nevezik

[5] Bimagic square

[6] Trimagic square

[7] Staircase method

14. fejezet - Képernyőkezeléssel kapcsolatos feladatok (szerző: Hernyák Zoltán)

Tartalom

14.1. feladat (Csillagok mindenhol – *szint: 3*). A 80×24 konzol képernyőre rajzoljunk * (csillag) karaktereket véletlenszerű pozíciókra (a legalsó sort technikai okokból hagyjuk üresen)! A program akkor álljon le, amikor minden pozícióra került * karakter!

Magyarázat: A megoldás egyszerűnek tűnik, az $x \in [0,79]$, az $y \in [0,23]$ koordinátákat véletlenszerűen lehet választani, majd az adott koordinátákra a *Console.SetCursorPosition* függvény segítségével lehet elmozgatni a kurzort, és kiírni a csillagot. Ebben a feladatban csak egyetlen nehézség van: ugyanazon koordinátákat többször is kisorsolhatjuk véletlenszerűen, ekképp $80 \cdot 24 = 1920$ csillag kiírásakor a képernyő még nincs teljesen tele. Lényegesen emelve a kiírt

darabszámot (mondjuk 8000 ismétlés után), a képernyő *valószínűleg* tele van.

Hogy eldönthessük, a képernyő teljesen tele van-e, szeretnénk kiolvasni a képernyő adott pozícióinak tartalmát, szeretnénk ellenőrizni, hogy minden karaktercellában csillag van-e. Erre nincs mód.

Helyette dupla adminisztrálást kell végeznünk. Egy – a képernyővel egyező méretű – 80×24-es mátrixot kell létrehoznunk. Érdemes ezt logikai típusra választani, így kiinduláskor *false* értékekkel lesz a mátrix tele. Amikor valamely *x,y* koordinátára csillagot írunk ki, akkor a logikai mátrix egyező koordinátájú cellájába helyezzünk *true* értéket! Ekkor a képernyő tele állapotának ellenőrzése már egyszerű.

> **14.2. feladat (Random technikai ellenőrzése – *szint: 3*).** Az előző feladat megoldása során megfigyelhetjük, hogy adott koordinátákat a véletlenszerű sorsolás többször is előállít. Számoljuk meg, mely koordinátát hányszor állított elő a sorsolás, amíg minden pozíció előállításra nem kerül (a csillagokkal tele a képernyő)! Adjuk meg, hogy a legtöbbet kisorsolt cella hányszor került sorra a véletlenszerű koordinátagenerálás közben! Adjuk meg a statisztikát, hogy melyik előfordulási darabszám hány cella esetén szerepelt!

Magyarázat: Az előző feladatban ismertetett megoldás során bemutatott nagy logikai mátrix helyett alkalmazzunk *int* típusú mátrixot, melyben meg tudjuk számolni, melyik cella hányszor került kisorsolásra. Az előfordulási darabszámokat egy listába kigyűjtve a feladat kérdéseire már könnyű válaszolni.

> **14.3. feladat (Csillagok mindenhol egyenletesen – *szint: 4*).** Érjük el, hogy az előző feladatban kirajzolásra kerülő csillag karakterek egyenletes sebességgel jelenjenek meg a képernyőn!. De ez ne illúzió legyen, és ne a *szerencsén* múljon, hanem a program legyen úgy megírva, hogy az garantáltan egyenletesen rakja ki a csillagokat a képernyőre! Az utolsó csillag kirakása után a program álljon le!

Magyarázat: Ha a koordináták sorsolása véletlenszerű, akkor gyakori az az eset, hogy olyan cellákat választunk ki, melyekbe már írtunk ki csillag karaktert. Ekkor nem garantálható az egyenletesség. A helyes gondolatmenet szerint gondolatban számozzuk be a cellákat 0-tól 1919-ig! Helyezzük el ezeket a számokat egy 1920 elemű vektorban, majd keverjük össze a számokat a vektoron belül! Így garantálható a megadott számok egyedisége a vektorban. Ha a vektor elemeit sorban kiolvassuk (feldolgozzuk), akkor a számok már véletlenszerű sorrendben kerülnek elő. Egy ilyen $n \in [0,1919]$ számhoz ki tudjuk számolni, mely x, y koordinátájú cellát azonosítja, s így sorban (de mégis véletlenszerű sorrendben) minden cellába csillagot írhatunk ki (14.1. forráskód).

> **14.4. feladat (Figyelemteszt – *szint: 3*).** A képernyőt osszuk fel függőlegesen három, vízszintesen két részre (ily módon összesen 6 kisebb téglalap alakú területre)! Minden területen más-más színt használunk (zöld, kék, sárga, piros, stb.). A képernyőre adott sebességgel (másodpercenként) villantsunk fel egy véletlenszerű pozíción egy csillag karaktert azzal a színnel, amely a terület jellemzője, ahova a véletlenszerűen választott pozíció esik! Villantsunk fel összesen 50 ilyen csillag karaktert! A felhasználónak meg kell adnia, szerinte melyik területre esett a legtöbb csillag karakter. A program döntse el, hogy igaz volt-e a válasz! Arra figyeljen a program, hogy garantáltan legyen ilyen terület, vagyis ha megvolt az 50 csillag, és két (vagy több) terület holtversenyben van, akkor a program néhány extra csillaggal toldja meg az 50-es darabszámot, amíg az egyértelműség kialakul!

Magyarázat: A képernyő vízszintesen 80 karakter széles. Ezt a szélességet három részre kell osztani, vagyis az egyes részek a $[0,26]$, $[27,53]$, $[54,79]$ közötti oszlopokat tartalmazzák. Függőlegesen 24 használható sor van, az egyes területek a $[0,11]$, $[12,23]$ közötti sorindexek. Ennek megfelelően, az *x* és *y* értékekre vonatkozó ellenőrzésekkel a területekbe besorolás egyszerűen elvégezhető.

Az egyes területekhez rendeljünk számlálókat, ehhez érdemes egy 6 elemű *int* vektort alkalmazni. Az egyes területekbe besorolás esetén az adott területhez tartozó számlálót növeljük 1-gyel. Az 50 villantás elérése után ellenőrizzük, hogy a maximális érték csak egyszer szerepel-e! Ha nem, akkor folytassuk még a villantásokat, amíg a maximális érték egyedivé válik! Érdemes az egyediségre kicsit erősíteni, mert ha az egyik területen mondjuk 10, a legtöbb villanást tartalmazó területen pedig 11 villantás volt, ez a különbség nehezen érzékelhető a program kezelője által. Előírhatjuk azt is, hogy a legtöbb villanást tartalmazó cellába legalább 30a különbség már szignifikáns.

A villantást a *Thread.Sleep(10)* várakozással tudjuk elérni: a képernyőre kiírjuk a csillagot, várakozunk a *Sleep* segítségével, majd letöröljük a csillagot.

> **14.5. feladat (Labirintus – *szint: 4*).** Egy text fájlban egy labirintust írunk le * és . karakterekekkel (* a fal, . a folyosó). Olvassuk be és jelenítsük meg a labirintust a képernyőn! Keressük meg, hol szerepel az elemek között az *S* karakter (start pozíció), és a *K* karakter (kijárat)! Az *S* karakterből pontosan egy szerepelhet, *K* kijáratból legalább egynek lennie kell. Határozzuk meg, hogy a *S* startpozícióból valamely kijáratig el lehet-e jutni! Mutassuk meg az utat!

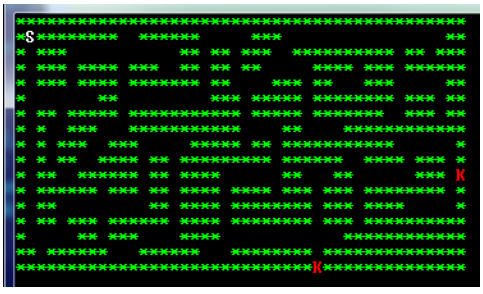
Magyarázat: A fájlból olvasásról már több helyen, elsőként a 9.13. feladat kapcsán írtunk. Hasonlóan elvégezhető a beolvasás (14.2. forráskód). A megjelenítés karakterenként történhet, így a különböző karaktereket más-más színnel jeleníthetjük meg (14.3. forráskód, 14.1. ábra).

Az *S* karakter pozíciójából kiinduló festéssel a labirintus elérhető celláit jelöljük meg (pont karaktert helyezünk ezekbe a cellákba). A festő algoritmust a 14.3. forráskód, a festés eredményét a 14.1. ábra tartalmazza.

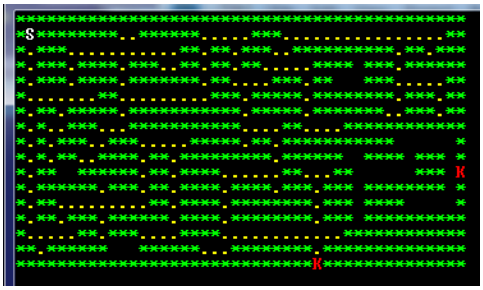
A festés után a kijáratok elérhetőségének ellenőrzése egyszerű: van-e olyan *K* karakter a mátrixban, amelynek 1 sugarú szomszédságában van pont karakter? Ha igen, akkor az *S*-ből ez a kijárat elérhető valamilyen útvonalon.

Ha az útra is szükségünk van, akkor módosítanunk kell a festő algoritmust. Ezen módszerrel ugyanis nem tudjuk meg az odavezető utat. Érdemes bevezetni egy másik, egyező méretű, de *int* típusú mátrixot, melyben induláskor legyenek 0 értékek! A befestés során bevezetjük a *távolság* fogalmát, mely az *S*-től mért lépések számát jelöli. Minden újabb festési lépés során a távolságot növeljük 1-gyel. Az *int* típusú mátrixban a pont karakterek elhelyezésekor beírjuk azok távolságát (a *K*-hoz az elérésekori távolságot). Az útvonal visszakeresése a *K*-ból indul ki. Ha egy cella távolsága *n* volt, akkor keressük, hol van a környezetében az *n - 1* távolságú cella (mert amikor festettünk, erről az *n - 1* távolságú celláról léptünk előre). A visszakeresést a 0 távolság elérésekor fejezzük be. Eközben az útvonalban részt vevő cellák tartalmát #-ra (hashmark) cseréljük. Az eredményt a 14.3. ábrán tekinthetjük meg. A feladathoz tartozó programrészletek a 14.2. ... a 14.6. forráskódokban tekinthetők meg.

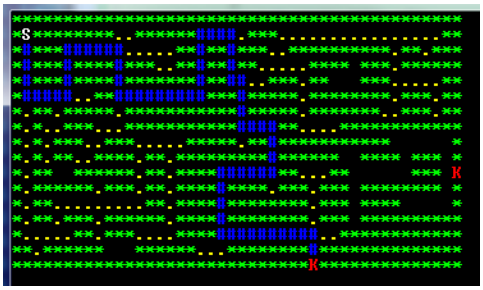
14.1. ábra. A labirintusalap (beolvasás utáni) megjelenítése



14.2. ábra. A labirintus befestése S-ből kiindulva



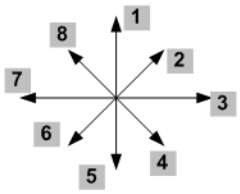
14.3. ábra. A menekülési útvonal



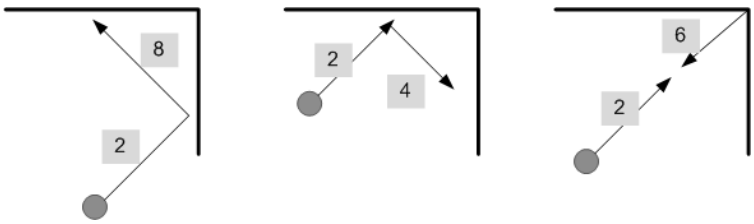
14.6. feladat (Falbontó – *szint: 3*). A képernyőre kerüljenek fel # (hashmark, kettőskereszt) jelek (vagy véletlenszerű helyekre, vagy olvassuk be a képernyő kezdőállapotát fájlból)! A képernyőn jelenjen meg egy pattogó labda (* karakter), mely 8 irányba képes pattanni! A labda pattanjon vissza a képernyő széléről (és a sarkokból is), valamint a # jelekről is! Figyeljünk arra, hogy a # jel sarkához érkezve sarokpattanást kell adni, míg lapjára érkezve lapról pattanás kell!

Magyarázat: A titok nyitja a folyamatos mozgás. Ha a labda elindult valamely irányba, akkor azt az irányt őrzi, amíg valami annak megváltoztatására nem kényszeríti. Kódoljuk be a mozgási irányokat például a 14.4. ábrán láthatóak szerint! Tároljuk a labda aktuális pozícióját (*x,y* koordináta), valamint az aktuális mozgási irányát a kódnak megfelelően! Ekkor könnyű kiszámolni, hogy a következő lépésben mi legyen az új koordinátája. Ha az új koordináta falat érne (képernyő széle, képernyőn belüli akadály), akkor módosítani kell a mozgási irányát (lásd például a 14.5. ábrának megfelelően).

14.4. ábra. Mozgási irányok kódjai



14.5. ábra. Irányváltozás 2-es mozgási irányból



Az akadályok (# karakterek) helyét koordináták formájában egy vektorban vagy listában tárolhatjuk el, esetleg alkalmazhatunk egy 80×25 (képernyő) méretű mátrixot, melynek megfelelő celláiban vagy a szóköz (cella üres), vagy a # (akadály) értékeket tárolhatjuk el.

14.7. feladat (Sok pattogó labda – *szint: 4*). A képernyőn jelenjen meg *N* darab pattogó labda, melyeket egy-egy * karakter szimbolizál! A labdák pattanjanak vissza a falról és a sarkokról, de egymáshoz ütközés esetén is pattanjanak vissza! A labdák számát, a labdák mozgási sebességét a program induláskor kérje be billentyűzetről!

Magyarázat: Az *N* darab labda esete nem különbözik sokban az előző feladatban ismertetett megoldási menettől. Az *N* labdához *N* koordináta és *N* mozgási irány tartozik. A labdák sorban veszik fel az új koordinátájukat (ha lehet), illetve pattannak vissza akár egymásról is. Az akadályok körét így bővíteni kell: nemcsak a falról, valamint a # karakterekkel jelölt akadályokról tud visszapattanni a labda, hanem akkor is, ha az új koordinátán egy másik labda „áll” éppen.

Ügyelni kell arra, hogy két összeütköző labda kölcsönösen módosítja egymás mozgási irányát (mindkét labdának módosul az aktuális iránya).

14.8. feladat (Almaevő csiga – *szint: 3*). Ismert játék, ahogy a képernyőn elindul egy csiga (@ karakterekből) valamely irányban. A csiga kezdetben csak 3 karakter hosszú, melyből a *fej* más színű. A képernyőn almák (zöld színű * karakterek) bukkannak fel, melyeket a csiga *megehet*. Ez akkor következik be, ha a *fej* ezen pozícióra lépne rá. Minden alma megevésekor a csiga hossza nő 1-gyel. A csiga győztesen kerül ki a játékból, ha mérete eléri az előre beállított értéket (pl. a 10-es hosszat). Az almák csak rövid ideig maradnak a képernyő adott pontján, a csigának emiatt csak kevés ideje van, hogy odaérjen. A csiga veszít, ha adott időn belül a mérete nem éri el a kívánt mértéket, vagy a fej egy, a csiga testét alkotó @ karakteren haladna át (a csiga saját magába harap).

A program működését vezérlő paraméterek értéke (idők, méretek, almák száma stb.) konstansok formájában szerepeljen a programban, hogy a működést könnyedén meg lehessen változtatni! A csigát a kurzornyilakkal kell irányítani.

Magyarázat: Egy *n* egységből álló csiga pontosan úgy viselkedik, mint egy *n* labdából álló sor. A labda mozgási irányának megfelelően kell kiszámolni a fej új koordinátáját, a fej mögötti csiga egység pedig felveszi a fej korábbi pozícióját és mozgási irányát, a harmadik egység a második egység régi koordinátáját és mozgási

irányát, stb. Ebből a szemszögből nézve a probléma már kezelhető, inkább időigényes, mint algoritmikusan nehéz.

14.9. feladat (Pingpong – *szint: 3*). A képernyőn egy pingponglabda pattogjon, az alsó részen egy 8 karakternyi széles (# karakterből álló) pingpongütő mozogjon vízszintesen! A feladat: az ütőt a lefele eső labda alá kell irányítani, hogy visszapattanjon róla. Ha ez nem sikerül, akkor a labda leesik. A program a labdát véletlenszerű pozícióról indítsa el alapvetően felfele átlós irányba, az ütő pedig álljon be a képernyő közepére, hogy minél több idő legyen az elején az ütőt az első leesés időpontjára helyzetbe állítani!

Magyarázat: A pattogó labda, amely visszapattan az akadályokról, már ismerős probléma. Ebben a feladatban csak az „akadály” mozgatása a gond. Egyetlen technikai problémát kell megoldani: a *Console.Readkey()* függvény szükséges az ütő mozgatásának lekérdezéséhez. Ez a függvény azonban úgy viselkedik, hogy ha nincs leütött billentyű éppen, akkor megvárja, míg azt leütik. Ez megengedhetetlen, mivel a labda esése eközben nem állhat le.

Ezt a problémát úgy orvosolhatjuk, hogy a *Console.KeyAvailable* propertyvel ellenőrizzük le, hogy van-e éppen leütött billentyű, mely esetben nyugodtan használhatjuk a ReadKey-t, nem fogja megakasztani a futás folyamatosságát.

```
ConsoleKeyInfo k = new ConsoleKeyInfo();
if (Console.KeyAvailable)
   k = Console.ReadKey();
switch (k.Key)
{
   case ConsoleKey.LeftArrow:
      // ...
      break;
}
```

14.10. feladat (Faltenisz – *szint: 3*). A játék nagyon hasonló az előző pingpongos feladathoz, de két ütő van, a képernyő két oldalán, a labda pedig a képernyő alsó széléről visszapattan, viszont a két szélén képes kirepülni. Ezt szeretné a két ütő a két oldalon megakadályozni, melyek függőlegesen mozognak. A labda az ütőkről visszapattan.

Egy játékos üzemmódban a két oldalon a két ütő szinkronban mozog, egy játékos képes mindkét ütőt irányítani. Két játékos esetén külön irányíthatjuk a bal oldali, és külön a jobb oldali ütőt.

Magyarázat: Az előző probléma megoldása után ez a feladat már jól kezelhető mennyiségű problémát tartalmaz. Ügyeljünk, hogy a két játékos üzemmódban a billentyűzeten egymástól nagy távolságra lévő 2-2 billentyűt jelöljük ki a kezeléshez, hogy elférjenek!

Bevezető információk: Életjáték: a négyzetrács mezőit celláknak, a korongokat sejteknek nevezzük. Egy cella környezete a hozzá legközelebb eső 8 mező (tehát a cellához képest *átlósan* elhelyezkedő cellákat is figyelembe vesszük, feltesszük, hogy a négyzetrácsnak nincs széle). Egy sejt/cella szomszédai a környezetében lévő sejtek. A játék körökre osztott, a kezdő állapotban tetszőleges számú (egy vagy több) cellába sejteket helyezünk. Ezt követően a játékosnak nincs beleszólása a játékmenetbe. Egy sejttel (cellával) egy körben a következő három dolog történhet:

- a sejt túléli a kört, ha két vagy három szomszédja van,
- a sejt elpusztul, ha kettőnél kevesebb (elszigetelődés) vagy háromnál több (túlnépesedés) szomszédja van,
- új sejt születik minden olyan cellában, melynek környezetében pontosan három sejt található.

Fontos, hogy a változások csak a kör végén következnek be, tehát az *elhalálozók* nem akadályozzák a születést és a túlélést (legalábbis az adott körben), és a születések nem mentik meg az *elhalálozókat*. A gyakorlatban ezért a következő lépéseket célszerű ilyen sorrendben végrehajtani:

- az elhaló sejtek megjelölése,
- a születő sejtek elhelyezése,
- a megjelölt sejtek eltávolítása.

14.11. feladat (Életjáték – *szint: 4*). Olvassuk be egy $N \times M$ méretű élettérmátrix kezdő állapotát, ahol a . (pont) karakter jelöli a cella üres állapotát, az # karakter jelöli hogy a mátrix adott cellájában élő sejt van! A kezdőállapotot jelenítsük meg a képernyőn, az üres cellákat szürke pont, a élő sejtes cellákat zöld színű # karakter jelölje! Futtassuk le a szimulációt, majd jelenítsük meg az élettér új állapotát! Folytassuk a szimulációt, amíg az <esc> billentyű leütésével ki nem lépünk belőle!

Magyarázat: A megoldás során érdemes két egyforma mátrixot kezelni. Az első mátrix tartalmazza az aktuális állapotot, a második mátrixban generáljuk a következő állapotot (az újonnan született cellákat máris berakjuk, az ezen körben elhalálozottakat eleve nem rakjuk át). A kijelzés a labirintus feladat kapcsán ismertetett módon is megvalósítható (14.6. ábra).

14.6. ábra. Az életjáték képernyője

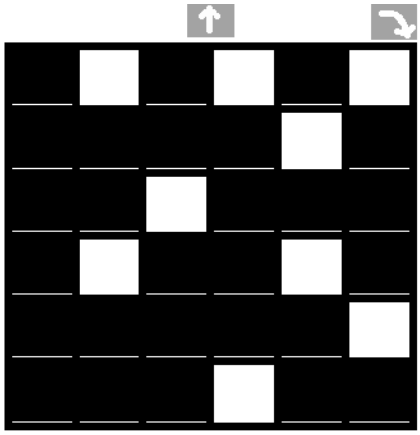
14.12. feladat (Digitális óra – *szint: 4*). Egy digitális órán megjelenítjük a óra, perc, másodperc értékeket két számjegyes alakban – emiatt összesen 6 számoszlop van. Az egyes számjegyek értékét digitálisan jelenítjük meg 4 LED segítségével, a LED-ek egymás alatt függőlegesen jelennek meg! A 4 LED elég, mivel 1 számjegy $[0 \ldots 9]$ értékeket vehet csak fel. A másodpercek, percek és órák számjegyeit más-más színnel jelenítsük meg! Az *on* állapotú LED-et egy #, az *off* állapotú LED-et pedig egy – jellel helyettesítsük! A program induláskor olvassa be az óra aktuális állapotát leíró mátrixot egy fájlból, ahol 4 sor van, soronként 6-6 jel, és az előbb leírtak szerint # és – jeleket tartalmaz! A beolvasás után a program állapítsa meg, hogy az valós időt ír-e le, majd jelezze ki a LED-eket, és másodpercenként frissítse az időkijelzést a mátrixban leírt időhöz képest indítva a számolást!

Magyarázat: Érdemes elkészíteni egy függvényt, amely egyetlen számjegyet ír ki a képernyőre. Ehhez kettes számrendszerbeli (pontosan 4 számjegyre kiegészítve) alakjából lehet kiindulni. Egy két számjegyű (pl. az órák száma) szám kiírása ezen függvény felhasználásával már egyszerű, csak a 2 számjegyet kell elválasztani egymástól (egyik a 10-zel való osztási maradék, a másik a 10-zel való egész osztás eredménye). Az órák, percek, másodpercek kiírása külön-külön, a 2 számot kiírni tudó függvény segítségével történhet meg. Az érintett forráskódok a 14.11. ... 14.13. forráskódokban olvashatóak.

14.7. ábra. A digitális óra képernyője

Bevezető információk: Verne Gyula *Sándor Mátyás* c. könyvében bemutat egy titkosítási módszert, melynek lényege, hogy egy $N \times N$ méretű mátrix celláit egy papírlapra rajzolták rá, majd a papírlapot adott celláknál kilyukasztották. Az így kapott maszk által szabad cellákba beírták a titkos üzenet első betűit, cellánként 1 betűt. Aztán elforgatták a maszkot, és az így kapott szabad cellákba beírták az üzenet következő betűit. Még kétszer forgatva és ismételve az eljárást a megfelelő hosszú üzenet minden karaktere bekerült az ily módon generált mátrix valamely cellájába. A maszk ismeretében az üzenet könnyedén dekódolható.

14.8. ábra. A *Sándor Mátyás*-titkosítás maszkja



14.9. ábra. A dekódolandó szöveg

R	H	G	A	A	Z			L	K	A	E	E	N			S	L	Ő	Ő	É	Z
Ü	Y	G	G	R	É			N	Y	E	Ő	L	M			Z	K	B	N	E	T
A	F	X	S	G	M			S	N	E	Ő	O	L			E	A	K	Z	L	D
N	T	L	Á	R	E			T	K	E	Z	K	Y			S	R	N	L	É	E
E	Z	L	F	T	É			G	A	G	E	A	E			I	Á	I	M	R	L
S	E	R	É	O	G			E	N	R	G	É	L			N	T	E	Ő	Z	N

14.13. feladat (Sándor Mátyás mátrixa – *szint: 4*). Egy fájlban a . (pont) és # (hashmark) karakterekből alkotott sorok írnak le egy $N \times N$ mátrixot oly módon, hogy *N* sora van, soronként *N* pont vagy hashmark karakter. Olvassuk be ezt a mátrixot egy $N \times N$ méretű logikai típusú mátrixba, ahol a pont karakter a *false*, a hashmark karakter a *true* értéket képviseli! A mátrix ebben az alakban egy *Sándor Mátyás*-féle titkosító maszkot ír le, a *true* jelzi, hogy azon a pozíción a mátrix cellája ki van vágva, a *false* a nem kivágott cella jele. A program ellenőrizze le, hogy a mátrix alkalmas-e üzenet kódolására (vagyis a mátrixot négyszer körbeforgatva minden cella fölé kerül kivágott cella a maszkból, és egyetlen cella fölé sem kerül egy kivágott cella sem kétszer vagy többször)!

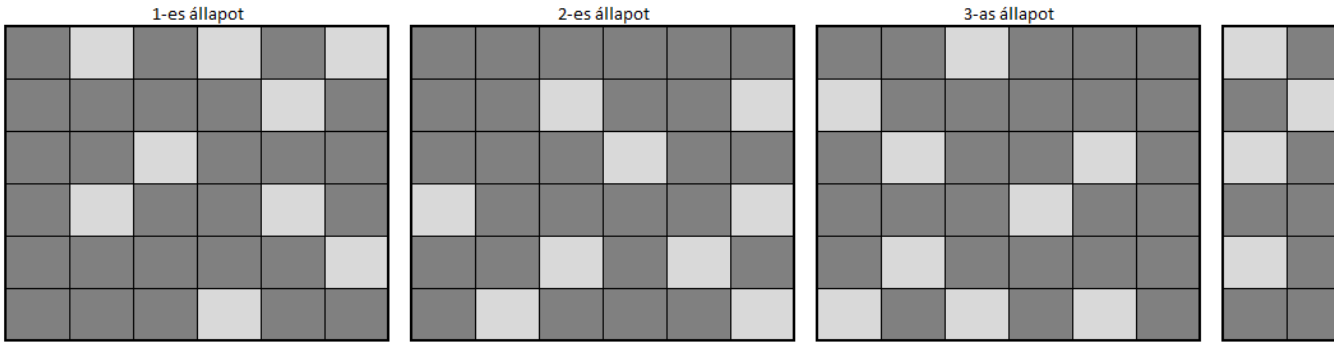
Magyarázat: Első feltétel: az *N* értéke páros kell, hogy legyen. A páratlan *N* esetén ugyanis van olyan cella, amely a mátrix kellős közepén helyezkedik el, és a forgatás közben helyben marad.

A forgatás során a mátrix valamely *i,j* koordinátájához hozzá kell rendelni a forgatás utáni *i',j'* koordinátát. A hozzárendelési összefüggések némi gondolkodással előállíthatóak:

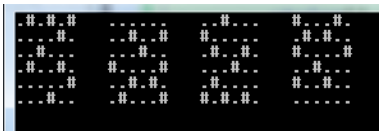
- j' = N - i - 1
- i' = (j + N) % N

A Verne-könyvben ismertetett mátrix elforgatásának fázisait a 14.10. ábra mutatja, az említett összefüggést alkalmazó program kimeneti képernyője pedig a 14.11. ábrán látható. A probléma megoldásához le kell generálnunk mind a 4 fázist, majd egy egyező méretű *int* mátrixot használva minden # elemhez tartozó cella értékét növelni 1-gyel (megszámoljuk, hányszor került az adott cella kitakarásra). A megszámlálás eredménye alapján a feladat könnyen megválaszolható: minden cellában 1 szerepel?

14.10. ábra. A mátrix a 4 elforgatási állapotban



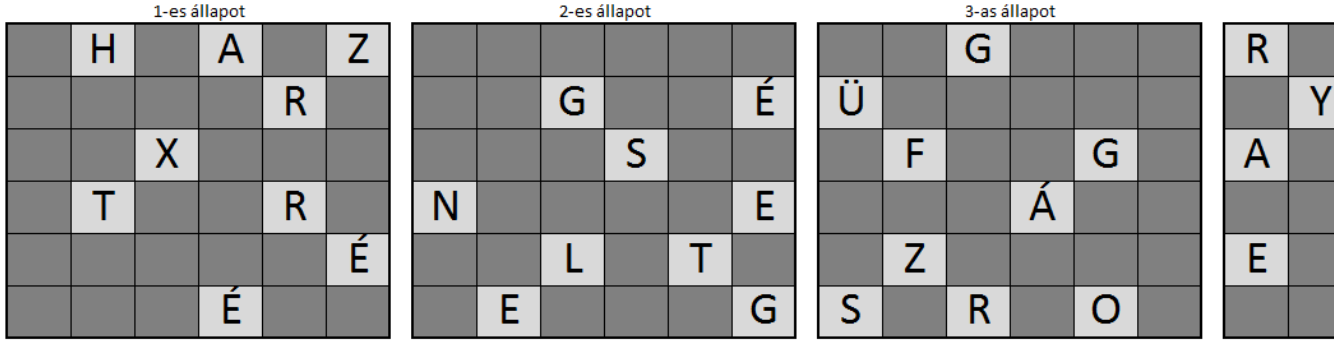
14.11. ábra. A mátrix elforgatását kijelző program képernyője



14.14. feladat (Sándor Mátyás-dekódoló – *szint: 4*). Az előző feladatban ismertetett módon olvassunk be egy *Sándor Mátyás*-féle maszkot, majd egy $N \times N$ karakterből álló titkos üzenetet! A program készítse el az üzenet titkosított alakját a maszk használatával!

Magyarázat: Soronként, balról jobbra haladva ki kell olvasni azokat a karaktereket az $N \times N$ méretű karaktermátrixból, amelyek fölött a maszk # kerül. Majd elforgatjuk a maszkot, és megismételjük a kiolvasást. Ezt kell ismételni, míg mind a 4 elforgatási fázisban kiolvastuk a kitakart karaktereket.

14.12. ábra. A kitakart betűk a 4 elforgatási állapotban



A kialakult szöveg: „HAZRXTRÉÉ GÉSNELTEG GÜFGÁZSRO RAYGAMLEF”. Esetünkben, a Verne-könyvben ezt a szöveget visszafele kell olvasni, és az adott történelmi korban (1867. év, az 1848-as szabadságharc utáni időszakban) értelmezhető. A teljes üzenet a könyv főszereplőjének, Sándor Mátyás grófnak szóló, a lázadást szító csoport egy titkos üzenete. A teljes szöveg generálásához mindhárom karaktermátrixra alkalmazni kell a maszkot, a maszk mind a négy elfogatási fázisában. A három szöveges mátrix dekódolva:

- HAZRXTRÉÉGÉSNELTEGGÜFGÁZSRORAYGAMLEF
- KENLEKKEGEMÖTYGANLANNOZAERÉLEJŐSLEGE
- LŐZEKRÉLŐBTZSEIRTNÖZALLÁNEZSÉKNEDNIM

Amelyet visszafele olvasva ezt kapjuk: *MINDEN KÉSZEN ÁLL. AZ ÖN TRIESZTBŐL ÉRKEZŐ LEGELSŐ JELÉRE AZONNAL NAGY TÖMEGEK KELNEK FEL MAGYARORSZÁG FÜGGETLENSÉGÉÉRT. XRZAH.* A végén álló értelmezhetetlen öt karakter az üzenetet küldő személy titkos aláírása.

14.15. feladat (Sándor Mátyás-kódoló – *szint: 4*). Az előző feladatban ismertetett módon olvassunk be egy *Sándor Mátyás*-féle maszkot, majd egy valahány karakterből álló titkos üzenetet! A program készítse el az üzenet titkosított alakját a maszk használatával! Az üzenet titkosítása során vegyük figyelembe, hogy az üzenet rövidebb vagy hosszabb is lehet, mint $N \times N$ karakter! A hosszabb üzenet esetén több mint egy titkosított mátrix generálódik. Amennyiben az üzenet vagy az üzenet egy része már rövidebb lenne, mint az $N \times N$ méret, úgy az üzenetet egészítsük ki random karakterekkel megfelelő méretre, és úgy kódoljuk le!

Magyarázat: Az előző feladat lényegében visszafele végrehajtva. A maszk elforgatása után a karaktermátrix kitakart celláiba be kell írni a szöveg soron következő betűit. A 4 elforgatás után egy karakterekkel feltöltött, teli mátrixot kell kapnunk. Azért kell a szöveget esetleg kiegészíteni, hogy az utolsó kódolt szöveget tartalmazó karaktermátrixban se legyen „kimaradt”, üres cella.

14.16. feladat (Sándor Mátyás-dekódoló – *szint: 4*). Az előző feladatokban ismertetett módon olvassunk be egy *Sándor Mátyás*-féle maszkot, majd egy egyező $N \times N$ méretű karakterekből (betűk, számok) álló mátrixot is! Ezen karaktermátrix szintén *N* sort, és soronként *N* karaktert tartalmaz. A dekódolómaszk segítségével készítsük el a titkosított szöveg dekódolását, és írjuk ki a képernyőre! A program jelezze ki, ha a dekódolómaszk felépítése nem megfelelő, vagyis a karaktermátrixban nem minden karakter került sorra, vagy valamelyik karakter többször is sorra került!

Magyarázat: A 14.14. feladat megoldásában előállt módszerből kell kiindulni. A beolvasás kicsit bonyolultabb, mert nem 1, hanem több dekódolandó karaktermátrixunk van, de mindegyikre ugyanúgy kell végrehajtani a dekódolást, tehát a feladat nem tartalmaz lényeges nehezítést az eredeti dekódolásos feladathoz képest.

A fejezet forráskódjai

14.1. forráskód. Képernyő feltöltése csillag karakterekkel egyenletes sebességgel

```
int[] v = new int[1920];
// feltöltés
for(int i=0;i<v.Length;i++)
   v[i] = i;
// összekeverés
Random rnd = new Random();
for(int i=0;i<v.Length*5;i++)
{
   int k = rnd.Next(0, v.Length);
   int l = rnd.Next(0, v.Length);
   int x = v[k];
   v[k] = v[l];
   v[l] = x;
}
// megjelenítés
foreach (int n in v)
{
   int x = n % 80;
   int y = n / 80;
   Console.SetCursorPosition(x, y);
   Console.Write("*");
}
// kesz
Console.SetCursorPosition(37, 12);
Console.Write(" KESZ ");
```

14.2. forráskód. Labirintus beolvasása

```
// static char[,] M = new char[25, 80];

StreamReader f =
 new StreamReader("labiritnus-1.txt", Encoding.Default);
int S = 0;
while (f.EndOfStream == false)
{
   string s = f.ReadLine();
   for (int j = 0; j < s.Length; j++)
      M[S, j] = s[j];
   S++;
}
f.Close();
```

14.3. forráskód. Labirintus megjelenítése

```
static void kiir()
{
   Console.SetCursorPosition(0, 0);
   for (int i = 0; i < 24; i++)
   {
      for (int j = 0; j < 79; j++)
      {
         switch (M[i, j])
         {
            case '*':
              Console.ForegroundColor = ConsoleColor.Green;
              break;
            case 'S':
              Console.ForegroundColor = ConsoleColor.White;
              break;
            case 'K':
              Console.ForegroundColor = ConsoleColor.Red;
              break;
            case '.':
              Console.ForegroundColor = ConsoleColor.Yellow;
              break;
            case '#':
              Console.ForegroundColor = ConsoleColor.Blue;
              break;
         }
         Console.Write(M[i, j]);
      }
      Console.WriteLine();
   }
}
```

14.4. forráskód. A befestő algoritmus

```
static void befest(int x, int y)
{
   if (M[y,x] != ' ') return;
   M[y,x] = '.';
   befest(x, y - 1);
   befest(x, y + 1);
   befest(x+1, y );
   befest(x-1, y );
}
```

14.5. forráskód. A módosított befestő algoritmus

```
static void befest(int x, int y, int tavolsag)
{
   if (M[y, x] == 'K') L[y,x] = tavolsag;
   if (M[y,x] != ' ' && M[y,x] != 'S') return;
   if (M[y, x] == ' ')
```

```
   {
      M[y, x] = '.';
      L[y, x] = tavolsag;
   }
   befest(x, y - 1,tavolsag+1);
   befest(x, y + 1,tavolsag+1);
   befest(x + 1, y, tavolsag + 1);
   befest(x - 1, y, tavolsag + 1);
}
```

14.6. forráskód. Az útvonal visszakeresése

```
static void utkeres(int x, int y)
{
   int tav = L[y, x];
   if (tav == 0) return;
   if (L[y - 1, x] == tav - 1)
   {
      M[y - 1, x] = '#';
      utkeres(x, y - 1);
   }
   else if (L[y + 1, x] == tav - 1)
   {
    M[y + 1, x] = '#';
    utkeres(x, y + 1);
   }
   else if (L[y, x-1] == tav - 1)
   {
    M[y, x-1] = '#';
    utkeres(x-1, y );
   }
   else if (L[y, x + 1] == tav - 1)
   {
    M[y, x + 1] = '#';
    utkeres(x + 1, y);
   }
}
```

14.7. forráskód. Az életjáték főprogramja

```
const int N = 20;
const int M = 60;
char[,] T = new char[N, M];
char[,] T2 = new char[N, M];
for (int i = 0; i < N; i++)
   for (int j = 0; j < M; j++)
      T[i, j] = '.';
//
StreamReader f =
   new StreamReader(@"c:\feladatok\eletjatek-1.txt", Encoding.Default);
int S = 0;
while (f.EndOfStream == false)
{
   string s = f.ReadLine();
   for (int j = 0; j < s.Length; j++)
      T[S, j] = s[j];
   S++;
}
f.Close();
//
while (true)
{
   kiir(T, N, M);
   Thread.Sleep(200);
   kalkulacio(T, T2, N, M);
   if (Console.KeyAvailable)
      if (Console.ReadKey(false).Key == ConsoleKey.Escape)
         break;
}
```

14.8. forráskód. A cellaszomszédok megszámolása

```
static int szomszedokSzama(char[,] T, int N, int M, int i, int j)
{
   int db = 0;
   if (i > 0 && T[i - 1, j] != '.') db++;
   if (i < N-1 && T[i + 1, j] != '.') db++;
   if (j > 0 && T[i , j-1] != '.') db++;
   if (j <M-1 && T[i, j + 1] != '.') db++;
   //
   if (i > 0 && j>0 && T[i - 1, j-1] != '.') db++;
   if (i < N - 1 && j>0 && T[i + 1, j-1] != '.') db++;
   if (i > 0 && j < M-1 && T[i - 1, j + 1] != '.') db++;
   if (i < N - 1 && j < M-1 && T[i + 1, j+1] != '.') db++;
   return db;
}
```

14.9. forráskód. A következő állapot kalkulációja

```
static void kalkulacio(char[,] T, char[,] uj, int N, int M)
{
   for(int i=0;i<N;i++)
      for (int j = 0; j < M; j++)
      {
         uj[i, j] = T[i, j];
         int db = szomszedokSzama(T, N, M, i, j);
         if (T[i, j] == '#')
         {
            if (db < 2 || db > 3) uj[i, j] = '.';
```

```
            }
            else if (T[i, j] == '.')
            {
               if (db == 3) uj[i, j] = '#';
            }
         }
      //
      for (int i = 0; i < N; i++)
         for (int j = 0; j < M; j++)
            T[i, j] = uj[i, j];
}
```

14.10. forráskód. Az aktuális állapot megjelenítése

```
static void kiir(char[,] T, int N, int M)
{
   Console.SetCursorPosition(0, 0);
   for (int i = 0; i < N; i++)
   {
      for (int j = 0; j < M; j++)
      {
         switch (T[i, j])
         {
            case '#':
              Console.ForegroundColor = ConsoleColor.Green;
              break;
            default :
              Console.ForegroundColor = ConsoleColor.Gray;
              break;
         }
         Console.Write(T[i, j]);
      }
      Console.WriteLine();
   }
}
```

14.11. forráskód. A digitális óra főprogramja

```
while (true)
{
   DateTime n = DateTime.Now;
   kijelez2(n.Hour, 3, 3, ConsoleColor.Green);
   kijelez2(n.Minute, 6, 3, ConsoleColor.Blue);
   kijelez2(n.Second, 9, 3, ConsoleColor.Yellow);
   Console.SetCursorPosition(3, 1);
   Console.ForegroundColor = ConsoleColor.Gray;
   Console.Write("{0,2}:{1,2}:{2,2}", n.Hour, n.Minute, n.Second);
   Thread.Sleep(1000);
}
```

14.12. forráskód. Két számjegy kijelzése

```
static void kijelez2(int szam, int x, int y, ConsoleColor szin)
{
   int a = szam % 10;
   int b = szam / 10;
   kijelez(x, y, b, szin);
   kijelez(x+1, y, a, szin);
}
```

14.13. forráskód. Egy számjegy kijelzése

```
static void kijelez(int x, int y, int szamjegy, ConsoleColor szin)
{
   for (int i = 0; i < 4; i++)
   {
      int bit = szamjegy % 2;
      szamjegy = szamjegy / 2;
      Console.SetCursorPosition(x,y+3-i);
      Console.ForegroundColor = szin;
      if (bit == 1) Console.Write('#');
      else Console.Write('.');
   }
}
```

15. fejezet - Listák feltöltésével kapcsolatos feladatok (szerző: Hernyák zoltán)

Tartalom

Az alábbi feladatokban egyszerű, C# alaptípusból alkotott listákkal, majd rekordokat tartalmazó listák feltöltésével kapcsolatos feladatokkal foglalkozunk.

> **15.1. feladat (Feltöltés billentyűzetről N elemszámra – *szint: 1*).** Egy törtszámokat tartalmazó listát töltsünk fel billentyűzetről oly módon, hogy a program kezelője a program elején megadja, hogy hány elemre lehet számítani(*n*), majd beírja az *n* számú törtértéket! A program a bevitel közben mindig írja ki, hogy hányadik számnál tart a bevitel, és mennyi van még hátra! A program a bevitel végén írja vissza az adatokat a képernyőre, a számokat egymás mellé írva, szóközzel elválasztva, majd adja meg a számok átlagát!

Magyarázat: A feladat egy egyszerű ciklussal megoldható, lényegében problémamentes. Ha valaki jobb minőségű kódot szeretne készíteni, legfeljebb arra kell ügyelnie, hogy a beírt szám valóban tört alakú legyen. Tudni kell, hogy *magyar területi beállítások* mellett a *Double.Parse* a törtszámokban a tizedesvessző megjelenésére számít, és nem a tizedespontra. A bekérésben esetleg előforduló tizedespontot vesszőre tudjuk cserélni a *Replace* metódus hívásával (a 15.1. forráskód).

> **15.2. feladat (Feltöltés billentyűzetről szám végjelig – *szint: 1*).** Egy törtszámokat tartalmazó listát töltsünk fel billentyűzetről oly módon, hogy a

program kezelője a program elején nem adja meg, hogy hány adat lesz! Helyette választunk egy speciális számértéket, például a 0.0 értéket. Amennyiben ezt a számot írnánk be, úgy a program fejezze be a bevitelt! A program a bevitel végén írja vissza az adatokat a képernyőre, a számokat egymás mellé írva, szóközzel elválasztva, majd adja meg a számok átlagát!

Magyarázat: A vége jel kezelése egy középen tesztelős ciklussal a leglátványosabb, legegyszerűbb. Ismert tény, hogy sokan ódzkodnak a középen tesztelős ciklustól, a *break* használatától. Megpróbálhatjuk megírni ezt a ciklust is akár elöl tesztelős, akár hátul tesztelős tisztán logikai ciklussal, amelyben nincs *break*, de azt fogjuk találni, hogy a kilépési tesztet `x==0.0` kétszer is be kell írnunk. Ekkor pedig kitörhet a lázadás a kódredundancia oldaláról. Nem törünk lándzsát egyik megoldás mellett sem, és ellene sem foglalunk állást (egyfajta megoldást lásd a 15.2. forráskódban).

15.3. feladat (Feltöltés billentyűzetről string végjelig – *szint: 1*). Egy törtszámokat tartalmazó listát töltsünk fel billentyűzetről oly módon, hogy a program kezelője a számok bevitelét egy speciális szöveggel zárja (pl. a „*” karaktert írja be, vagy begépeli, hogy „vége”)! A program a bevitel végén írja vissza az adatokat a képernyőre, a számokat egymás mellé írva, szóközzel elválasztva, majd adja meg a számok átlagát!

Magyarázat: Az előző feladat megoldása után ez sem jelenthet gondot, mindössze arra kell ügyelni, hogy a vége ellenőrzést a *double.Parse* előtt hajtsuk végre, mivel a különleges stringértékek nem parzolhatóak át számmá, kivétel dobásával leállna a program (lásd a 15.3. forráskód).

15.4. feladat (Feltöltés véletlen számokkal – *szint: 1*). Egy listát töltsünk fel véletlen egész számokkal oly módon, hogy a páros sorszámú listaelemek a $[10 \dots 50]$, a páratlan sorszámú listaelemek a $[40 \dots 80]$ intervallumból kerüljenek ki! A listába kerüljön be *n* ilyen szám, az *n* értékét a program induláskor kérje be! A program a végén írja vissza az adatokat a képernyőre, a számokat egymás mellé írva, vesszővel elválasztva, majd adja meg a számok átlagát!

Magyarázat: Gyakori elvi hiba kezdő programozóknál, hogy nem egy, de több *Random* példányt is készítenek programjaikban. Tudnunk kell, hogy a *Random* példányok a véletlenszám-generáláshoz szükséges kezdőértéket a rendszerórából veszik át. Ezért a *közel egy időben* készült példányok egy kezdőértékről indulnak, ugyanazon véletlen számokat fogják előállítani (természetesen ha ugyanazon intervallumot használjuk). Különböző intervallumok használata sem indokolja a több *Random* példány jelenlétét a programban, mivel minden egyes véletlenszám-előállítás esetén külön-külön kell megadni a kívánt intervallumot. Ezért is egyetlen Random példányt érdemes használni végig a generálás során (15.4. forráskód).

Másrészről ügyelnünk kell az átlagszámításra, mivel itt már nem törtszámok, de egész számok átlagáról van szó. Ekkor a korábban használt `sum/n` nem fog helyes eredményt adni, ha a *sum* típusa (és az *n* típusa is) egész szám. Nem érdemes sem a *sum*, sem az *n* típusát másra választani, helyette explicit típuskonverziót kell használni az osztás elvégzésekor `(double)sum/n` (lásd 15.5. forráskód).

15.5. feladat (Feltöltés véletlen növekvő számokkal – *szint: 1*). Egy listát töltsünk fel véletlen egész számokkal oly módon, hogy az első listaelem a $[10 \dots 30]$ intervallumba essen, a következő listalemek az őket megelőző elemtől legyenek „valamennyivel” nagyobbak, a különbség az $[1 \dots 5]$ intervallumból kerüljön ki! A lista hosszát (*n* darab) a program induláskor kérje be! A program a végén írja vissza az adatokat a képernyőre, a számokat egymás mellé írva, vesszővel elválasztva, majd adja meg a számok átlagát!

Magyarázat: A lista első elemét megadott módon kell képezni. A következő elem értékének előállításához az előző értékből kell kiindulni, melyhez újabb random értéket kell adni. A feladat nem túl bonyolult, de eredménye nagyon értékes. Anélkül kaphatunk „rendezett”, véletlen számokat tartalmazó számsort, hogy rendezőalgoritmust kellene ismernünk! A megoldást lásd a 15.6. forráskódban.

15.6. feladat (Feltöltés fájlból (a) – *szint: 1*). Egy text fájl soronként egy törtszámot tartalmaz. Írjunk olyan programot, amely bekéri a fájl nevét, majd beolvassa a számokat a fájlból! A bevitel véget ér, ha a fájl minden sorát beolvastuk, vagy üres sort olvastunk be a fájlból. A program a bevitel végén írja vissza az adatokat a képernyőre, a számokat egymás mellé írva, vesszővel elválasztva, majd adja meg a számok átlagát!

Magyarázat: A fájlból beolvasást többször bemutattuk már. Mivel ennek során stringeket olvasunk be, az üres sor detektálása sem lehet problémás. Az üres sor felfedezéséhez egy jó minőségű megoldást mutatunk be a 15.7. forráskódban. Alkalmazzuk a beolvasott sorra a *Trim* függvényt (ez levágja a sor bevezető és záró white-space karaktereit), a kapott string hosszát vessük össze a 0-val!

15.7. feladat (Feltöltés fájlból (b) – *szint: 1*). Egy text fájl soronként egy vagy több törtszámot tartalmaz. Amikor több szám is szerepel egy sorban, akkor vesszővel vannak elválasztva. A vessző után szóközök is szerepelhetnek a jobb vizuális tördelés érdekében. Írjunk olyan programot, amely bekéri a fájl nevét, majd beolvassa a számokat a fájlból! A bevitel véget ér, ha a fájl minden sorát beolvastuk, vagy üres sort olvastunk be a fájlból. A program a bevitel végén írja vissza az adatokat a képernyőre, a számokat egymás mellé írva, vesszővel elválasztva, majd adja meg a számok átlagát!

Magyarázat: A beolvasott értékes sort a *Split* függvény segítségével vághatjuk fel a vessző karakter mentén törtszámokra. A darabokra alkalmazzuk a *Trim* függvényt, hogy a feladatban is említett felesleges szóközöket levágjuk. A kapott számokat adjuk hozzá a listához (15.8. forráskód)!

15.8. feladat (Feltöltés billentyűzetről listaszerűen – *szint: 1*). Egy egész számokat tartalmazó listát töltsünk fel billentyűzetről oly módon, hogy a program kezelője egy időben egyszerre több számot is beírhat, vesszővel elválasztva! Ekkor minden számot adjunk hozzá a listához! Ha csak egy számot írna be a kezelő, akkor azt az egy számot. A bevitelt akkor fejezzük be, ha a kezelő egyetlen számot sem ír be (üres string)! A program a bevitel végén írja vissza az adatokat a képernyőre, a számokat egymás mellé írva, vesszővel elválasztva, majd adja meg a számok átlagát!

Magyarázat: Az előző feladatban ismertetett módszer tökéletesen alkalmas ebben a feladatban is, csak a sor beolvasását ez esetben nem a fájlból, hanem billentyűzetről kell elvégezni (15.9. forráskód).

15.9. feladat (Feltöltés billentyűzetről összeghatárig – *szint: 1*). Egy egész számokat tartalmazó listát töltsünk fel billentyűzetről oly módon, hogy a program kezelője addig írhat be számokat, amíg azok összege el nem éri az előre megadott értéket (például 100)! A program a bevitel közben folyamatosan figyelje az összeghatárt, illetve mindig írja ki, hogy hányadik számnál tart a bevitel, hol tart az összeg, mennyi van még hátra az értékhatárig! A program a bevitel végén írja vissza az adatokat a képernyőre, a számokat egymás mellé írva, vesszővel elválasztva, majd adja meg a számok átlagát!

Magyarázat: A korábban ismertetett adatbekéréseket alapul vehetjük, mindössze a kilépés feltétele változik meg ez esetben (15.10. forráskód).

15.10. feladat (Feltöltés egyedi számokkal – *szint: 1*). Egy egész számokat tartalmazó listát töltsünk fel billentyűzetről oly módon, hogy a listába nem kerülhet be ugyanazon szám többször is! Ha a program kezelője olyan számot írna be, amely már volt, akkor a program ezt jelezze ki! A bevitelt a kezelő akkor fejezheti be, ha sikerült neki egymás után háromszor is olyan értéket beírni, amely még nem szerepelt korábban (amit a program elfogad). A program a bevitel végén írja vissza az adatokat a képernyőre, a számokat egymás mellé írva, vesszővel elválasztva, majd adja meg a számok átlagát!

Magyarázat: Olvassuk el figyelmesen a feladatot! Nem akkor kell befejezni az adatbevitelt, amikor a lista mérete eléri a 3-at (3 különböző számot tartalmaz), hanem amikor egymás után 3-szor sikeres adatbevitel történt. Érdemes bevezetni egy *sikeres* számlálót, melyet minden siker esetén növelünk 1-gyel, ha hibázunk, akkor lenullázzuk. A bevitel akkor fejeződik be, amikor a számláló eléri a 3-at.

A sikeresség ellenőrzéséhez érdemes kifejleszteni egy egyszerű függvényt, mely megadja a lista aktuális tartalma és egy új *x* érték esetén, hogy ez az *x* érték szerepel-e a listán vagy sem (15.11. forráskód). Ez alapján a főprogram már könnyen kialakítható (15.12. forráskód).

15.11. feladat (Fájlból két halmaz – *szint: 2*). Egy text fájl soronként egy vagy több törtszámot tartalmaz. Amikor több szám is szerepel egy sorban, akkor

vesszővel vannak elválasztva. A vessző után szóközök is szerepelhetnek a jobb vizuális tördelés érdekében. A text fájl két számlistát tartalmaz, a két számlista között egy üres sor szerepel a fájlban. A program olvassa be mindkét számlistát két különböző listaváltozóba! Adjuk meg, melyik számlistában szereplő számoknak nagyobb az átlaga!

Magyarázat: A 15.7. feladat megoldása során bemutatott módszer alapján a feladat már szinte problémamentesen megoldható. Az üres sor első elérésekor nem kell leállni a beolvasáskor, hanem folytatni kell a második üres sor vagy a fájl végének eléréséig.

Trükkös megoldást alkalmazhatunk, ha jól uraljuk a referencia típust. Egy harmadik lista változót vezethetünk be, mely először az első listára mutat, majd váltáskor a második listára állunk át vele. Számoljuk az üres sorok számát is, de csak akkor lépünk ki a ciklusból, ha ez a számláló eléri a 2-t. A megoldást lásd a 15.13. forráskódban.

A fejezet forráskódjai

15.1. forráskód. Lista feltöltése billentyűzetről

```
List<double> szamok = new List<double>();
Console.Write("Hany szamrol lesz szo:");
int n = int.Parse(Console.ReadLine());
for (int i = 0; i < n; i++)
{
   Console.Write("Kerem a {0}. szamot:", i + 1);
   string s = Console.ReadLine();
   double d = double.Parse( s.Replace('.',','));
   szamok.Add(d);
}
//
foreach (double x in szamok)
   Console.Write("{0} ", x);
Console.WriteLine();
//
double sum = 0;
foreach (double x in szamok)
   sum = sum+x;
Console.WriteLine("Atlag={0}", sum / n);
```

15.2. forráskód. Lista feltöltése végjelig

```
List<double> szamok = new List<double>();
int i = 0;
while (true)
{
   Console.Write("Kerem a {0}. szamot:", i + 1);
   string s = Console.ReadLine();
   double d = double.Parse( s.Replace('.',','));
   if (d == 0.0) break;
   else szamok.Add(d);
   i++;
}
```

15.3. forráskód. Lista feltöltése string végjelig

```
List<double> szamok = new List<double>();
int i = 0;
while (true)
{
   Console.Write("Kerem a {0}. szamot:", i + 1);
   string s = Console.ReadLine();
   if (s == "*" || s == "vege") break;
   double d = double.Parse( s.Replace('.',','));
   szamok.Add(d);
   i++;
}
```

15.4. forráskód. Lista generálása

```
List<double> szamok = new List<double>();
Random rnd = new Random();
for(int i=0;i<n;i++)
{
   if (i % 2 == 0) szamok.Add(rnd.Next(10, 51));
   else szamok.Add(rnd.Next(40, 81));
}
```

15.5. forráskód. Egész számok átlaga

```
int sum = 0;
foreach (int x in szamok)
   sum = sum+x;
Console.WriteLine("Atlag={0}", (double) sum / n);
```

15.6. forráskód. Rendezett lista generálása

```
List<int> szamok = new List<int>();
Random rnd = new Random();
int x = rnd.Next(10, 31);
for(int i=0;i<n;i++)
{
```

```
   szamok.Add(x);
   x = x + rnd.Next(1, 6);
}
```

15.7. forráskód. Lista feltöltése számokkal file-ból

```
List<double> szamok = new List<double>();
StreamReader f = new StreamReader(@"szamok.txt", Encoding.Default);
while (f.EndOfStream == false)
{
   string s = f.ReadLine();
   if (s.Trim().Length==0) break;
   double d = double.Parse( s.Replace('.',',') );
   szamok.Add( d );
}
f.Close();
```

15.8. forráskód. A fájlban egy sorban több szám is szerepelhet

```
List<double> szamok = new List<double>();
StreamReader f = new StreamReader(@"szamok.txt", Encoding.Default);
while (f.EndOfStream == false)
{
   string s = f.ReadLine();
   if (s.Trim().Length==0) break;
   string[] ss = s.Split(',');
   foreach(string p in ss)
   {
      double d = double.Parse( p.Trim().Replace('.',','));
      szamok.Add( d );
   }
}
f.Close();
```

15.9. forráskód. A beírt sorban több szám is szerepelhet

```
List<int> szamok = new List<int>();
while (true)
{
   string s = Console.ReadLine();
   if (s.Trim().Length==0) break;
   string[] ss = s.Split(',');
   foreach(string p in ss)
   {
      int d = int.Parse(p.Trim());
      szamok.Add( d );
   }
}
```

15.10. forráskód. Adatbekérés összeghatárig

```
List<int> szamok = new List<int>();
int sum = 0;
while (sum<100)
{
   string s = Console.ReadLine();
   if (s.Trim().Length==0) break;
   int d = int.Parse(s.Trim());
   szamok.Add( d );
   sum = sum + d;
}
```

15.11. forráskód. Ellenőrző függvény: az *x* szerepel-e a listán

```
static bool szerepel_e(List<int> l, int x)
{
   foreach (int d in l)
      if (d == x) return true;
   //
   return false;
}
```

15.12. forráskód. Az adatbekérés főprogramja

```
List<int> szamok = new List<int>();
int siker = 0;
while (siker<3)
{
   string s = Console.ReadLine();
   int d = int.Parse(s.Trim());
   if (szerepel_e(szamok, d) == true) siker = 0;
   else
   {
      szamok.Add(d);
      siker++;
   }
}
```

15.13. forráskód. Két számhalmaz beolvasása

```
List<double> szamok1 = new List<double>();
List<double> szamok2 = new List<double>();
List<double> akt = szamok1;
StreamReader f = new StreamReader(@"szamok.txt", Encoding.Default);
int ures_db = 0;
while (f.EndOfStream == false)
{
   string s = f.ReadLine();
   if (s.Trim().Length == 0)
   {
      ures_db++;
      if (ures_db == 2) break;
      akt = szamok2;
      continue;
   }
   string[] ss = s.Split(',');
   foreach (string p in ss)
   {
      double d = double.Parse(p.Trim().Replace('.', ','));
      akt.Add(d);
   }
}
f.Close();
```

16. fejezet - Listákkal kapcsolatos feladatok (szerző: Hernyák Zoltán)

Tartalom

A fejezet forráskódjai

A feladatok alapszintű listafeldolgozással megoldhatóak. Ennek során kell egy vagy több lista, melynek feltöltése nem feltétlenül fontos része a feladatoknak. A megoldások során a kiinduló adatokat tartalmazó listákat általában nem kell módosítani, új listákat kell előállítani.

> **16.1. feladat (Maximum – *szint: 2*).** Olvassuk be egy lista tartalmát billentyűzetről valamely, az előző feladatban megadott módszer szerint! Kérjünk be egy újabb értéket (*A*), mely érték a program kezelője „szerint" a listabeli számok maximuma! A program ellenőrizze, hogy ez valóban a maximum-e!

Magyarázat: Óvatosan lássunk az ellenőrzésnek: nem elég, hogy a listában nem találunk az *A* értéknél nagyobb számot, az *A* értéknek szerepelnie kell a listabeli számok között is! Persze nekiláthatunk a megoldásnak oly módon is, hogy meghatározzuk a lista kalkulált maximumát, és összevetjük az *A* értékkel – de az kevésbé izgalmas (lásd 16.1. forráskód).

> **16.2. feladat (Páros számok maximuma – *szint: 3*).** Olvassuk be egy lista tartalmát billentyűzetről valamely, az előző feladatban megadott módszer szerint! A program adja meg a listában szereplő páros számok közül a legnagyobb számértéket (ügyeljünk arra az esetre, mikor a listában minden szám páratlan)!

Magyarázat: A maximumkeresés egyszerű feladat, ha tudjuk a megfelelő kezdőértéket. A kezdőértéknek a lista első elemét szoktuk választani, mely választás egyúttal akár a végeredmény, a maximum is lehet. De ez esetben nem tehetjük, mivel nem lehetünk biztosak abban, hogy az első elem páros-e. Azt sem tudhatjuk, hogy a második páros-e. Lehet, hogy egyik sem páros, és nem találunk kezdőértéket.

Meg kellene keresnünk az első páros számot, kiválasztani mint kezdőértéket, majd folytatni a keresést. Ez igazából két ciklust igényel, megoldása a 16.2. forráskódban látható. Kissé erőforrás-pazarlóbb, de algoritmikusan jóval áttekinthetőbb megoldás, ha a lista páros elemeit átmásoljuk egy második listába. A továbbiakban ezen lista elemszáma alapján azonnal megállapítható, hogy van-e egyáltalán páros szám, illetve könnyű megkeresni a maximumot (lásd a 16.3. forráskódot).

> **16.3. feladat (Unió, metszet – *szint: 2*).** A 15.7. feladatban ismertetett módon olvassunk be egyetlen fájlból két számlistát (*A* és *B*)! Fogjuk fel az egyes számlistákat mint egy-egy számhalmazt! Generáljuk le az $A \cup B$, $A \cap B$ számhalmazokat, és írjuk ki az értékeket a képernyőre! Ügyeljünk arra, hogy az unió és metszet listákban ne szerepeljen egyetlen szám sem kétszer!

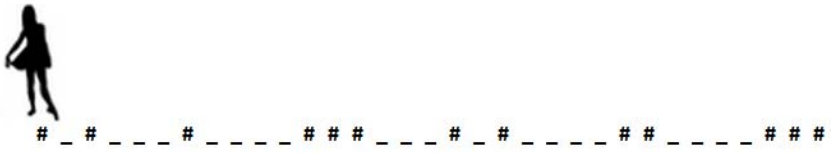
Magyarázat: Ez az egyszerű alapalgoritmusok használatát jelenti, speciálisan listára kialakítva. Hogy kissé érdekesebb legyen a megoldás, írjuk át olyan függvényekre, melyek a két listát paraméterként megkapva a generált listát adják meg visszatérési értékként! Mindkét részfeladat igényli a listabeli számok egyediségét, így érdemes a 15.11. forráskódban már bemutatott egyediség-ellenőrző függvényt alkalmazni.

Az *unio* művelet képzése nagyon egyszerű: mindkét listából szükséges minden elemet befogadni, ami még nem szerepelt. A megoldást lásd a 16.4. forráskódban. A *metszet* kicsit bonyolultabb ellenőrzési mechanizmusú: olyan elemeket keresünk, amelyek szerepelnek az *A* és *B* listában, de nem szerepelnek még a metszetben (16.5. forráskód).

> **16.4. feladat (Erdei ösvény – *szint: 3*).** Egy erdei ösvényen vizes pocsolyák és száraz szakaszok váltják egymást. Szimbolizálja a szárazföldet a numerikus 1, a vizes részt a 2 érték! Töltsünk fel egy listát véletlenszerű 1 és 2 értékekkel oly módon, hogy nagyobb valószínűséggel kerüljön bele víz, mint szárazföld (például 70%-30% arányban)! Legyen a lista első és utolsó eleme garantáltan 1, vagyis szárazföld! A kész listát jelenítsük meg a képernyőn oly módon, hogy a szárazföldet a # (hashmark), a vizet a _ (aláhúzás) karakterrel jelöljük! A konkrét arányokat a program kérje be billentyűzetről!
>
> Az erdei ösvény egyik végén áll Piroska, és szeretne átjutni az ösvény másik végére a nagymamához. Piroska *n* hosszú pocsolyás szakaszt képes átugorni (*n* értékét kérjük be billentyűzetről). A program generálja le a listát adott arányokkal, jelenítse meg a képernyőn, majd adja meg, hogy Piroska át tud-e kelni az ösvényen száraz lábbal (16.1. ábra)!

Magyarázat: A feltöltést adott valószínűséggel a geometriai valószínűségek szerinti módszerrel oldhatjuk meg. Sorsoljunk véletlen számot $[1,100]$ intervallumban! Ha az kisebb, mint 30egyenlő, akkor víz. A megjelenítést is egyszerű megoldani a korábbi feladatok alapján.

16.1. ábra. Piroska és az ösvény



Piroska átkelési problémája során érdemes bevezetni egy számlálót, melyben a szomszédos víz elemek számát számoljuk. Ha a lista következő eleme víz – növeljük a számlálót. Ha szárazföld, akkor lenullázzuk. Ekképpen már csak a számláló maximális értékét kell meghatároznunk (a 16.6. forráskód).

16.5. feladat (Szigetek – *szint: 3*). Tegyük fel, hogy Amerika partjaitól elindul egy repülő Európa partjai felé! A repülő egyenes vonalban egyenletes sebességgel repül végig adott magasságban. A repülés során a felszín magasságát méri, melyet rögzít. A számértékek méterben értendők, és egy listába kerülnek be. A víz felszínén mért magasságérték mindig 0, és negatív magasságot nem lehet mérni. A pozitív értékek a víz fölé kiemelkedő szárazföldet mutatnak. A lista első és utolsó értéke értelemszerűen pozitív szám (Amerika partvidékéről indulunk, és Európa partvidékének elérésekor áll le a mérés). A két pont között vízfelszíni és szárazföldi szakaszok váltják egymást.

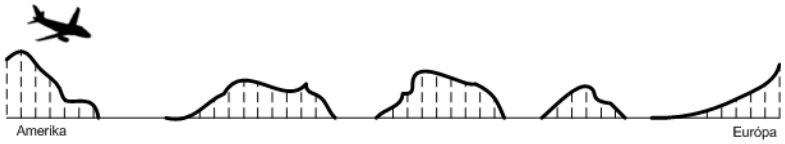
A program töltse fel a listát véletlen értékekkel, de a feladat szövegében foglaltaknak megfelelően! Vagyis ne egyszerű nulla és nem nulla értékek váltsák egymást véletlenszerűen, hanem ha vízfelszíni szakasz kezdődik, akkor az folyamatos legyen, valamint a szárazföldi szakasz is felismerhető legyen! A szárazföldi szakaszok mindig egy-egy szigetet képviselnek (16.2. ábra).

A program határozza meg egy feltöltött lista alapján:

- hány sziget található Amerika és Európa között,
- hanyadik sorszámú szigeten található a legmagasabb pont (hegycsúcs),
- milyen hosszú a leghosszabb sziget (egységekben)!

Magyarázat: Ügyeljünk a szélsőséges esetekre: elképzelhető, hogy Amerika és Európa között nincs vizes szakasz (egybefüggő szárazföldet alkot). Másik eset: nincs sziget, a két pont között egybefüggő vizes szakasz terül el. A legmagasabb hegycsúcs esetén elképzelhető, hogy ezen maximális magasság ugyanazon szigeten belül többször vagy több szigeten is előfordulhat. Szintén ügyeljünk arra, hogy a legmagasabb pont ne Európa vagy Amerika partvidékéhez tartozzon, hanem valamely szigeten forduljon elő!

16.2. ábra. A repülőgép útvonala



A lista feltöltése úgy történhet, hogy kisorsoljuk, hogy víz vagy szárazföld következzen, majd a szakasz hosszát. Ezek után megfelelő mennyiségű 0-t vagy nem 0-t helyezünk el a listában. Ügyeljünk rá, hogy a lista első és utolsó eleme ne 0 legyen! A program maradék részét úgy kell megírnunk, hogy ne függjön a feltöltési mechanizmusunktól (ne építsen semmilyen ott szerzett tudásra)!

Ezek után a piroskás feladatban ismertetett módon meg kell számolni a 0 szakaszok hosszát. Valahányszor 0-ról nem 0-ra váltunk, sziget kezdődik (kivéve az utolsó ilyet, ahol Európa kezdődik). Ügyeljünk rá, hogy ha nem volt, csak 1 ilyen váltás, akkor nincs sziget a két kontinens között! Ha 0 ilyen váltás volt, akkor összefüggő szárazföld van.

A leghosszabb sziget meghatározásához a Piroska leghosszabb vizes szakaszának meghatározásánál leírtakból kell kiindulni. A maximális pont keresése sem problémás, mivel tudjuk, hogy egyik pont sem alacsonyabb 0 méternél, így a maximumkeresés kiinduló értéke lehet a 0. Mivel meg kell számolnunk, hány sziget van, nem nehéz a maximum megtalálásakor feljegyezni a sziget sorszámát is.

A fejezet forráskódjai

16.1. forráskód. A lista maximumának ellenőrzése

```
// List<int> L = new List<int>();
bool nagyobb_volt = false;
bool szerepelt = false;
foreach (int x in L)
{
   if (x == A) szerepelt = true;
   if (x > A) nagyobb_volt = true;
}
if (nagyobb_volt == false && szerepelt == true)
   Console.WriteLine("eltalalta a maximumot");
else
   Console.WriteLine("nem talalta el");
```

16.2. forráskód. A legnagyobb páros szám keresése

```
// List<int> L = new List<int>();
int i = 0;
while (i < L.Count && L[i] % 2 != 0)
   i++;
if (i < L.Count)
{
   int max = L[i];
   for (int j = i + 1; j < L.Count; j++)
      if (L[j] % 2 == 0 && L[j] > max)
         max = L[j];
   Console.WriteLine("A paros max={0}", max);
}
else Console.WriteLine("Nincs paros eleme a listanak");
```

16.3. forráskód. Egyszerűbb algoritmus, több memórialekötés

```
// List<int> L = new List<int>();
List<int> lp = new List<int>();
foreach (int x in L)
   if (x % 2 == 0) lp.Add(x);
if (lp.Count == 0)
   Console.WriteLine("Nincs paros eleme");
else
{
   int max = lp[0];
```

```
   foreach (int x in lp)
      if (x > max) max = x;
   Console.WriteLine("Paros max = {0}", max);
}
```

16.4. forráskód. Az unió függvénye

```
static List<int> unio(List<int> A, List<int> B)
{
   List<int> ret = new List<int>();
   foreach (int x in A)
      if (szerepel_e(ret, x) == false)
         ret.Add(x);
   foreach (int x in B)
      if (szerepel_e(ret, x) == false)
         ret.Add(x);
   return ret;
}
```

16.5. forráskód. A metszet függvénye

```
static List<int> metszet(List<int> A, List<int> B)
{
   List<int> ret = new List<int>();
   foreach (int x in A)
      if (szerepel_e(ret, x) == false && szerepel_e(B,x)==true)
         ret.Add(x);
   return ret;
}
```

16.6. forráskód. A leghosszabb vizes szakasz hossza

```
static int leghosszabb(List<int> L)
{
   int db = 0;
   int max = 0;
   foreach (int x in L)
   {
      if (x == 2) db++;
      else db = 0;
      if (db > max) max = db;
   }
   return max;
}
```

17. fejezet - Rekordok és listák együtt (szerző: Hernyák Zoltán)

Tartalom

A fejezet forráskódjai

A feladatok mini-adatbázisokon végzett szokásos műveletekkel foglalkoznak. A mini-adatbázist rekordokból épített listák reprezentálják. A feladatok akkor érdekesek, ha az adatok listája nem 3-5, hanem legalább 20 elemű. Mivel rekordokról van szó, ahol rekordonként 3-8 mező is előfordulhat, ez komoly mennyiségű adatbevitelt jelent, amit nem célszerű billentyűzetről megoldani. Érdemes tehát kialakítani egyfajta módszert, a listaelemek beolvasása fájlból történjen! A mátrixos és listás fájlfeltöltések megvalósítása után ez nem szabad, hogy nagy gondot okozzon.

A feladatok megfogalmazásában rekordszerkezetek kerülnek definiálásra. A rekordokat a mezők neveinek felsorolásával adjuk meg az egyszerűség kedvéért. A mezők nevei alapján a mezők típusa általában egyértelműen kitalálható, és ritkán fontos a feladatok szempontjából. Ahol mégis az, ott megadjuk. Előfordulhat, hogy személyekhez tartozó rekordok esetén szerepel a *neme* mező. Ez ekkor a személy nemét jelöli (férfi, nő). Ezt egyetlen karakteren tároljuk, 'F', ha férfi, 'N', ha nő. Ezenfelül gyakran használunk skálaértékeket (mint az első feladatban a *bulizási hajlam* mező. A skálaértékek egyfajta mérőszámok, $[0 \dots 10]$ közötti értékek, ahol a 0 az adott skála egyik vége, a 10 érték a másik végét jelképezi.

A rekordokat text fájlban tudjuk tárolni, leírni. Ez esetben egy sor egy rekordot ír le, az egyes mezők között egy olyan elválasztó karaktert használunk, mely nem fordul elő a mezőkbeli értékek leírása során. Erre mindenki választhat egy saját kedvenc karaktert, javasoljuk a függőleges vonal karaktert használni. Feltételezhetjük, hogy a fájl megfelelő szerkezetű, vagyis minden sorában megfelelő mennyiségű mező szerepel, és a tartalom a mezők típusához illeszthető. A beolvasás során ez természetesen ellenőrizhető. A nem megfelelő rekordok (sorok) eldobhatóak, de a beolvasás végén érdemes ez esetben ezt egy hibaüzenettel jelezni, megadva az eldobott sorok számát.

17.1. feladat (Beolvasás – *szint: 2*). Készítsünk egy rekordot a barátaink adatainak tárolásához *(név, születési dátum, neme, bulizási hajlam)*. Ez utóbbi mező egy skálaérték. Egy fájlban tároljuk a rekordokat, soronként egy rekordot! A program olvassa be a rekordokat, helyezze el egy listában! A beolvasás addig tart, amíg el nem érjük a fájl végét, vagy egy üres sorhoz nem ér. A program a beolvasott adatokat jelenítse meg táblázatos formában, soronként visszaírva azokat a képernyőre!

Magyarázat: A beolvasás során soronként egy rekordunk van, a mezők egymástól | (függőleges vonal) karakterrel vannak elválasztva. Egy sort olvasunk, a *Split* segítségével felbontjuk részekre, és bemásoljuk a rekord megfelelő mezőibe, típuskonverziókat végezve.

17.1. ábra. Az input fájl tartalma

```
new 1
1  Kiss Lajos|1970.04.07|F|8
2  Nagy Évike|1975.01.30|N|9
3  Fájó Ödönke|1980.08.12|F|2
4
```

A 17.1. forráskód mutatja be a barát rekord (csak mezőket tartalmazó osztály) deklarációját. A fájl beolvasása és feldolgozása a 17.2. forráskódban látható. Az üres sorokat is kiszűrjük a fájlból, a megfelelő típusbeli értékeket a megfelelő *Parse* függvénnyel állítjuk elő. A képernyőre írást (ellenőrzési szereppel) a 17.3. forráskód mutatja be. A dátum jelen esetben csak év, hónap, nap adatokat tartalmaz, a felhasznált *DateTime* ennél többet tud (óra, perc stb.). Ezért a kiírás során a dátumot formázó string segítségével (yyyy.MM.dd) kényszerítjük a formátum betartására.

> **17.2. feladat (Lapozásos megjelenítés – *szint: 2*).** Az előző feladatban beolvasott listát jelenítsük meg a képernyőn táblázatos, lapozható formában! A lapozást előre-hátra módon oldjuk meg, képernyőnként 20 rekord kiírással kalkulálva! A lapozást a PageUp és PageDown gombokkal lehessen megvalósítani! Ezenfelül a Home billentyűvel lehessen az első, az End billentyűvel az utolsó lapra ugorni!

Magyarázat: A lista beolvasását a 17.1. feladatban leírtak szerint végezhetjük el. Vezessünk be egy változót, mely annak sorszámát tartalmazza, hogy hanyadik elemtől kezdődik a kiírás (ez kezdetben az első barát, 0 sorszámtól indul)! Készítsünk egy kiíró függvényt, mely a lista adott kezdő indexétől kezdve ír ki 20 rekordot (ha van annyi egyáltalán hátra)! A billentyűzetkezeléssel kapcsolatosan a 9.11. feladat kapcsán már írtunk, így minden szükséges ismeret rendelkezésre áll, hogy megoldhassuk a problémát.

> **17.3. feladat (Buliszervezés (a) – *szint: 3*).** Az előző feladatban beolvasott lista alapján határozzuk meg, hogy van-e elég 20 évnél idősebb barátunk, akikkel egy születésnapi bulit tudunk összehozni! Ehhez olyan barátokra van szükségünk, akik bulizási hajlama legalább 5-ös skálaértékű. A buli minimális létszáma 10 fő. Amennyiben van elég barátunk, adjuk meg (soroljuk fel) a neveiket!

Magyarázat: Egyszerűen meg kell számolni azokat a barátokat a listában, akik életkor és bulizási hajlam mezőiben megfelelő értékek vannak. Ha ezeket a barátokat kiválogatjuk egy külön listába (ami a második rész, a képernyőre kiírás miatt szükséges), akkor a megszámlálással sem kell foglalkoznunk külön, mivel a lista elemszáma (*Count*) megadja ezt az információt.

Egyedüli probléma lehet a 20 éves életkor meghatározása, mivel az életkorok nem, csak a születési dátumok ismertek. A számítógépünk belső órájának aktuális dátumát a *DateTime.Now* módon kérdezhetjük le, mely egy teljes *DateTime* típusú érték, megadja a dátumot és az időpont értékét is. A két dátumot (aktuális, születési) kivonva egymásból megkapjuk a két időpont közötti különbséget (hány év, hónap, nap, óra, perc, másodperc távolságra esik a két időpont egymástól). Ez egy *TimeSpan* típusú érték, melynek a *.Days*, *.Hours* stb. nevű propertyjein keresztül tudjuk felmérni az időkülönbséget (17.2. ábra):

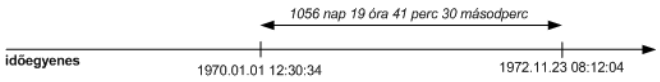
```
DateTime d1 = DateTime.Parse("1970.01.01 12:30:34");
DateTime d2 = DateTime.Parse("1972.11.23 08:12:04");
TimeSpan t = d2 - d1;
Console.WriteLine("{0} nap {1} {2} {3}",
   t.Days, t.Hours, t.Minutes, t.Seconds);
```

Sajnos ez nem könnyen járható út a 20 éves kor meghatározásához, bár a *t.Days* értéket ha osztjuk 365-tel, megkapjuk, (nagyjából) hány év telt el, s ily értelemben hány éves az adott napon a barátunk. Egyszerűbb azonban a jelenlegi dátum évéből kivonni 20 évet. Ha a születési dátum éve ugyanezen évre, vagy korábbi évre esik, akkor tekinthetjük a barátunkat legalább 20 évesnek.

```
int koraiEv = DateTime.Now.Year - 20;
barat p;
p.szuletesi_datum.Year<=koraiEv) ...;
```

A bulizó kedvű barátaink kiválogatását végző függvény kódja a 17.4. forráskódban olvasható.

17.2. ábra. A timespan értelmezése



> **17.4. feladat (Buliszervezés (b) – *szint: 3*).** Az előző feladatot egészítsük ki azzal, hogy a szervezendő buli maximális létszáma 15 fő! Ha több szóba jöhető barátunk lenne, akkor csak 15 nevet soroljunk fel! A feladat bonyolításaként ez esetben válogassuk ki a 15 bulizásra leghajlamosabb barátunk nevét!

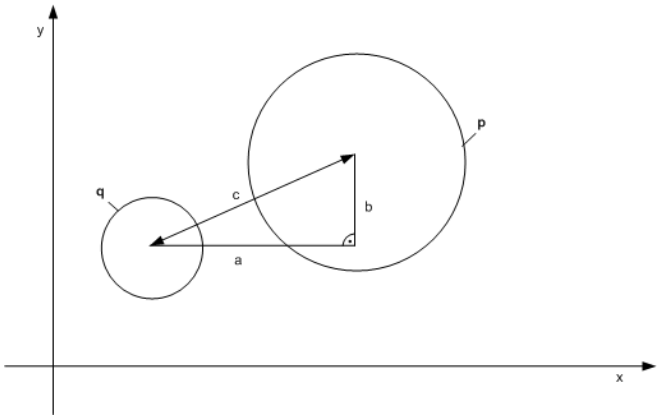
Magyarázat: Az előző feladatban megadott függvény képes kiválogatni a bulizásra alkalmas barátainkat. Ha a generált lista több mint 15 elemű, akkor a listát rendezni kell bulizási hajlam szerint (17.5. forráskód), majd venni az első 15 elemét. A nevek kiírásával a feladat kész.

> **17.5. feladat (Átfedő körök – *szint: 3*).** Descartes-koordinátarendszerbeli körök adatait tároljuk rekordban *(X,Y,sugár)* értékek formájában. A rekordok listáját töltsük fel fájlból! A program keresi arra a kérdésre a választ, hogy hány olyan kör található, amelynek nincs közös pontja egyetlen más körrel sem. Adjuk meg ezen körök számát, és soroljuk fel a középpontjuk koordinátáit is!

Magyarázat: Első lépésben két kör (*p* és *q*) középpontjainak távolságát kell meghatározni. A két kör középpontját kössük össze egy szakasszal (*c*), majd készítsünk egy derékszögű háromszöget, melynek ezen *c* szakasz az átfogója! Az *a* befogó értéke a két kör középpontjának *x* koordinátáinak különbsége, hasonlóan kell kiszámolni a *b* befogó értékét is. E kettő ismeretében az átfogó a *Pitagorasz-tétel* szerinti képlettel egyszerűen megkapható:

```
double a = p.x-q.x;
double b = p.y-q.y;
double c = Math.Sqrt(a*a+b*b);
```

17.3. ábra. A körök távolsága



Ha ismerjük a két középpont távolságát, akkor könnyű összevetni ezt az értéket a két sugár (p.sugar+q.sugar) összegével. Ha a *c* távolság nagyobb, akkor a körök távol vannak egymástól. Ha a kettő egyenlő, akkor a körök 1 ponton érintkeznek, ha a *c* a kisebb érték, akkor a körök „összeérnek", vagy akár egyik kör teljesen tartalmazza a másikat (a feladat szempontjából lényegtelen, melyik).

Ez alapján egyszerű készíteni egy olyan függvényt, mely két körrekord-paraméter esetén megmondja, hogy a köröknek van-e közös pontjuk, vagy sem (17.6. forráskód). A listában szereplő minden rekordot minden más rekorddal össze kell vetni, és megszámolni hány esetben ad meg a függvény *false* értéket.

17.6. feladat (Kollégiumi statisztika – *szint: 3*). Egy kollégium diákjairól rekordokban tárolunk adatokat *(név, életkor, neme, lakóhely távolsága km-ben, féléves tanulmányi átlaga, hány hónapja kollégista)*. Az adatokat olvassuk be fájlból, majd válaszoljunk az alábbi kérdésekre:

1. kik azok a diákok, akik tanulmányi átlaga nem éri el a 3,5 értéket, és messzebb laknak, mint 40 km,
2. mi a női és férfi diákok százalékos aránya,
3. mi a kollégiumbeli női és férfi diákok tanulmányi átlaga,
4. adjuk meg életkori bontásban, hány férfi és hány női diákunk van!

Magyarázat: Az *a)* rész egyszerű megszámolós feladat, nem tartalmaz nehézséget. A *b)* pont is megszámolós feladat, ahol a férfiak és nők darabszámát a teljes lista létszámához kell arányítani. A *c)* részben először összegezni kell a férfi diákok tanulmányi átlagait, majd osztani kell a férfiak számával. Ügyeljünk arra, hogy ha nincs férfi a listában, akkor nehogy a 0 darabszámmal osszunk! Hasonlóan járhatunk el a nők átlagának számításakor is.

A *d)* rész esetén úgy értelmezzük a feladatot, hogy adott évben születetteket létszámát kell összeszámolni. Jobb minőségű kód esetén érdemes a legkisebb (minimum) és legnagyobb (maximum) születési évet kikeresni, majd a kettő közötti éveket egy ciklussal sorba venni. Minden évhez könnyű megszámolni hány férfi (vagy nő) rekord tartozik. Ha nem akarjuk a minimum-maximum számítással húzni az időt, akkor lehetséges még az aktuális évből kiindulva visszafele mondjuk 100 évet vizsgálni. Amely évben 0 darabszám jön ki, azt nem kell kijelezni.

17.7. feladat (Zenegyűjtemény – *szint: 2*). Kis együttesünk 15 éve létezik. Az általa játszott zeneszámok adatait rekordokban írjuk le *(cím, komponálás éve, hossza másodpercben, népszerűségi mutatója)*. A kedveltségi mutató egy skálaérték. Az adatokat olvassuk be fájlból! Határozzuk meg:

1. ha 60 perces koncertet kellene adni, van-e elég zeneszámunk ehhez,
2. soroljuk fel a zeneszámok címeit kedveltség szerint csökkenő sorrendben (vagyis a népszerűbbekkel kezdjük),
3. határozzuk meg évenként, hogy melyik évben mennyi volt az adott évi nótáink átlagos népszerűségi rátája! Igaz-e, hogy minden évben sikerült ezen az átlagértéken javítani?

Magyarázat: Az *a)* kérdés egyszerűen megválaszolható: adjuk össze a zeneszámok hosszát! Ne feledjük, ez másodpercbeli érték lesz!

A *b)* kérdés sem tartalmaz semmi ravaszságot: rendezzük a listát népszerűség szerinti sorrendben (17.13. forráskód), majd írassuk ki a címeket!

A *c)* részhez gyűjtsük ki egy 15 elemű *double* vektorban az adott évben írt zeneszámok népszerűségi rátájának átlagát (ügyeljünk arra, hogy nem feltétlenül készült minden évben zeneszám, így lehet, hogy nem képezhető átlag)!

17.8. feladat (Sporteredmények – *szint: 4*). A focimeccsek eredményeit rekordokban tároljuk *(meccs dátuma, 1. csapat neve, rúgott gólok száma, 2. csapat neve, rúgott gólok száma)*. Az adatokat olvassuk be fájlból! Ha egy nyertes meccs esetén a győztes 3 pontot, a vesztes 1 pontot kap, döntetlen esetén 2-2 pontot kapnak, akkor

1. adjuk meg az egyes csapatok idény végén összegyűjtött pontszámait, pontszám szerint csökkenő sorrendben,
2. adjuk meg a legtöbb gólt rúgott és legtöbb gólt kapott csapat (csapatok) neveit,
3. soroljuk fel csapatonként dátum szerint csökkenő sorrendben az ellenfelek neveit, és a mérkőzés eredményét!

Magyarázat: Az *a)* feladat megoldásához érdemes készíteni egy újabb rekordot (csapat) *név, pontszám* mezőkkel. Készítsünk egy olyan listát, melybe az ilyen *győzelmek* rekordok szerepelnek (17.7. forráskód), oly módon, hogy a csapatnevek egyediek (a vizsgálat kódja a 17.8. forráskódban), és minden csapathoz ki vannak számolva a pontszámok (valamint a *b)* feladathoz készülve a rúgott és kapott gólok számai is – 17.9. forráskód)! A lista generálását a 17.10. forráskód vezérli. A listát pontszám szerint csökkenőbe rendezni már egyszerű feladat (17.11. forráskód).

A *b)* feladatra is tudunk válaszolni, ha az *a)* megoldása során a rúgott és kapott gólok számait is kiszámítjuk (ahogy azt korábban már jeleztük). A maximális értékek meghatározása után ki kell írni minden csapat nevét, amelyhez a maximális értékek (rúgott, kapott gólok száma) tartoznak.

A *c)* feladat megoldásához is ezen *csapatok* listából kell kiindulnunk, mivel abban a csapatok nevei már egyedileg szerepelnek. Az egyes csapatnévhez ki kell gyűjteni a meccseket a teljes listából (17.12. forráskód), majd rendezni a kigyűjtést dátum szerint csökkenő sorrendbe.

17.9. feladat (Autóválasztás – *szint: 3*). Egy cég telephelyén tárolt személyautók adatait rekordban írjuk le *(rendszám, kényelmi fokozat, fogyasztás, tank teljes űrtartalma, tankban lévő üzemanyag mennyisége, szállítható személyek száma)*. Az autók adatait olvassuk be fájlból, majd kérjünk be egy úti célt (hány km távolságra kellene menni), és hogy hány személyt kell eljuttatni oda! Írjuk ki, mely autókat javasoljuk az úthoz! Rendezzük ezt a listát költségek

szempontjából olyan módon, hogy:

1. kerüljenek előre azok az autók, amelyekbe nem kell tankolni indulás előtt, az aktuális tankolási állapotuk szerint így is elegendő az üzemanyaguk az úthoz! Ezen autók egymás közötti sorrendje legyen kényelmi fokozat szerint csökkenő!
2. következzenek azok az autók, amelyekbe tankolni kell valamennyit, de utána képesek lennének az utat megtenni! Ezen autók egymás közötti sorrendje legyen az, hogy mennyi üzemanyagra van még pluszban szükségük!
3. végül következzenek azok az autók, amelyeket hiába tankolnánk teljesen tele, akkor sem képesek a távot egyetlen tankolással megtenni (közben is meg kellene állni újra tankolni)! Ezen autók rendezettsége legyen a szükséges tankolási megállások száma szerinti csökkenő sorrendben!

Az elkészült lista alapján adjuk meg, mely autókat kell igénybe venni, hogy az adott számú személy elszállítása a célhoz megvalósítható legyen! Vegyük figyelembe, hogy autónként 1 sofőrt is biztosítani kell, ezzel csökkentve az adott autóban szállítható személyek számát!

Magyarázat: Az *a)* részfeladat megválaszolásához gyűjtsük ki a szóban forgó autókat külön listába! Az autók fogyasztása és a tankban lévő mennyiség alapján meghatározható a tankolás nélküli távolság, melyet össze kell vetni a úti cél távolságával. Ne feledjük, hogy a fogyasztás 100 km-re kerül megadásra (lásd alább)! A kiválogatott lista rendezését a korábbi feladatokban ismertetett példák szerint végezhetjük el.

```
// pl. 42 liter / 8 liter * 100 => 525 km elég a benzin
double megteheto_tav = p.tank/p.fogyasztas*100;
```

A *b)* feladathoz úgy lehet egyszerűen kiválogatni a részt vevő autókat, hogy az összes autóból kivesszük az *a)* feladatban kiválogatott autókat, majd sorra kiszámoljuk, melyikbe mennyi benzin kell még (a képletet lásd lejjebb). A rendezés miatt érdemes az autó rekordot ilyen mezővel kiegészíteni, mert a rendezés során ezeket az értékeket kell összehasonlítani. A *b)* feladatban csak azokat az autókat kell megtartani, amelyeknél a $p.tank + p.szukseges\_benzin \leq p.tank\_urtartalom$.

```
// pl. (520 km - 340 km) * 8 liter / 100 => 14.4 liter kell még bele
p.szukseges_benzin = (cel_tav-megteheto_tav)*p.fogyasztas*p.tank/100;
```

A *c)* feladathoz a maradék autókat kell kigyűjteni. A szükséges tankolások számának kiszámításakor nem feltétlenül kell úgy gondolkodni, hogy elsőként teletankoljuk az autót (ezt levonjuk a szükséges benzinből), majd elosztjuk és felfele kerekítjük a maradék benzinigényt és a tank méretének hányadosát. Ha csak kevés hiányzik a tele tankhoz, akkor ezzel akár +1 tankolási igényt is generálhatunk. Gondolkodjunk úgy, hogy amíg van benne benzin, addig megyünk a kezdő tankkal, és csak akkor állunk meg tankolni, amikor már 0-n áll a mutató! Tehát a szükséges tankolások száma a szükséges benzinmennyiség és a tank méretének hányadosa (szükség esetén felfele kerekítve ezt a hányadost). Mivel ezen szempont szerint kell rendeznünk, így erre is érdemes felvenni a rekordba egy mezőt.

A három részlista autóit újra érdemes összefűzni egyetlen nagy listává. Kezdjük el feldolgozni ezen újra összefűzött listát rekordról rekordra! Az autók szállítási kapacitását 1-gyel csökkentve vehetjük figyelembe a sofőrre vonatkozó megjegyzés miatt. Addig kell haladnunk, amíg elég sok autót soroltunk be a szállítási listába. Ha nem elég az autópark a szállítás megoldásához (lesz maradék el nem szállítható személy), akkor azt jeleznünk kell.

17.10. feladat (Órarendi ütközések – *szint: 3*). Egy iskolában lévő órákat rekordokkal írunk le *(osztály, tanár neve, tanterem, nap, óra sorszáma, hossza)*. Az óra sorszáma mutatja az óra kezdő időpontját (pl. 2. óra). A hossza azt mutatja, hogy egy óra (szimpla) vagy két óra (dupla óra) egyben, stb. Olvassuk be az adatokat fájlból! Határozzuk meg, az órarendben van-e ütközés az alábbi szempontok szerint:

1. ugyanazon tanárnak ugyanabban az időpontban (akár átfedéssel) van-e két különböző órája,
2. ugyanannak az osztálynak ugyanabban az időpontban (akár átfedéssel) van-e két különböző órája,
3. ugyanabban a teremben van-e egy időben több óra kiírva.

Ha találunk ütközéseket, soroljuk fel az ütközésbe került órák adatait!

Magyarázat: Az *a)* kérdés megválaszolásához minden rekordot minden rekorddal össze kell vetnünk (egymásba ágyazott foreach ciklusok), hogy lássuk, van-e ütközés két *p* és *q* órarendi rekord között. Ez akkor áll fenn, ha $p \neq q$, a $p.tanar = q.tanar$, $p.nap = q.nap$, és az óra sorszáma és hossza kölcsönösen nem fedik át egymást (a rekord deklarációja a 17.14. forráskódban, a két órarendi rekord ütközésének vizsgálata a 17.14. forráskódban, a teljes vizsgálat a 17.14. forráskódban olvasható).

A *b)* kérdés valójában teljesen hasonló az *a)* kérdéshez, csak nem a tanárra koncentrál, hanem az osztályra. Csakúgy, mint a *c)*, ahol pedig a terem egyezése az elsődleges szempont az átfedések vizsgálata során.

A fejezet forráskódjai

17.1. forráskód. A barát rekord deklarációja

```
class barat
{
   public string nev;
   public DateTime szuletesi_datum;
   public char neme;
   public int bulizasi_hajlam;
}
```

17.2. forráskód. Barátok beolvasása fájlból

```
List<barat> lista = new List<barat>();
StreamReader r = new StreamReader("baratok.txt", Encoding.Default);
while (r.EndOfStream == false)
{
   string s = r.ReadLine();
   if (s.Trim().Length == 0) continue;
   string[] ss = s.Split('|');
   barat b = new barat();
   b.nev = ss[0];
   b.szuletesi_datum = DateTime.Parse(ss[1]);
   b.neme = char.Parse(ss[2]);
   b.bulizasi_hajlam = int.Parse(ss[3]);
   lista.Add(b);
```

```
}
r.Close();
```

17.3. forráskód. Barátok kiolvasása

```
foreach (barat p in lista)
{
   Console.WriteLine("{0,-20} {1,10} {2,1} {3}",
      p.nev,
      p.szuletesi_datum.ToString("yyyy.MM.dd"),
      p.neme,
      p.bulizasi_hajlam);
}
```

17.4. forráskód. A bulizós barátaink kiválogatása

```
static List<barat> kivalogat(List<barat> mindenki)
{
   int koraiEv = DateTime.Now.Year - 20;
   List<barat> ret = new List<barat>();
   foreach (barat p in mindenki)
      if (p.szuletesi_datum.Year<=koraiEv && p.bulizasi_hajlam>=5)
         ret.Add(p);
   //
   return ret;
}
```

17.5. forráskód. Rendezés bulizási hajlam szerint

```
static void rendezes(List<barat> bulizok)
{
for(int i=0;i<bulizok.Count;i++)
 for(int j=i+1;j<bulizok.Count;j++)
  if (bulizok[i].bulizasi_hajlam > bulizok[j].bulizasi_hajlam)
  {
   barat c = bulizok[i];
   bulizok[i] = bulizok[j];
   bulizok[j] = c;
  }
}
```

17.6. forráskód. Van-e közös pontja a két körnek

```
static bool van_e_kozos_pont(kor p, kor q)
{
   double a = p.x - q.x;
   double b = p.y - q.y;
   double c = Math.Sqrt(a * a + b * b);
   double r = p.sugar + q.sugar;
   if (r > c) return true;
   else return false;
}
```

17.7. forráskód. A rekordok deklarációja

```
class meccs
{
   public DateTime meccs_datuma;
   public string csapat_1;
   public int rugott_gol_1;
   public string csapat_2;
   public int rugott_gol_2;
}

class csapat
{
   public string csapat_neve;
   public int pontszama;
   public int rugott_golok;
   public int kapott_golok;
}
```

17.8. forráskód. Az egyediség ellenőrzése

```
static csapat kikeres(string csapat_neve, List<csapat> csapatok)
{
   foreach (csapat p in csapatok)
      if (p.csapat_neve == csapat_neve)
         return p;
   //
   return null;
}
```

17.9. forráskód. Egy csapat rekord kezelése

```
static void csapat_gen(List<csapat> lista,
           string csnev, int rugott, int kapott)
{
   csapat r = kikeres(csnev, lista);
   if (r == null)
```

```
   {
      r = new csapat();
      r.csapat_neve = csnev;
      lista.Add(r);
   }
   // golok szama
   r.rugott_golok = r.rugott_golok + rugott;
   r.kapott_golok = r.kapott_golok + kapott;
   // pontszam
   if (rugott > kapott)
      r.pontszama = r.pontszama + 3;
   else if (rugott == kapott)
      r.pontszama = r.pontszama + 1;
}
```

17.10. forráskód. A csapatok lista generálása

```
static List<csapat> pontszamit(List<meccs> meccsek)
{
   List<csapat> ret = new List<csapat>();
   foreach (meccs p in meccsek)
   {
      csapat_gen(ret, p.csapat_1, p.rugott_gol_1, p.rugott_gol_2);
      csapat_gen(ret, p.csapat_2, p.rugott_gol_2, p.rugott_gol_1);
   }
   //
   return ret;
}
```

17.11. forráskód. A csapatok rendezése pontszám szerint csökkenőbe

```
static void rendezes(List<csapat> csapatok)
{
   for (int i = 0; i < csapatok.Count; i++)
      for (int j = i + 1; j < csapatok.Count; j++)
         if (csapatok[i].pontszama < csapatok[j].pontszama)
         {
            csapat c = csapatok[i];
            csapatok[i] = csapatok[j];
            csapatok[j] = c;
         }
}
```

17.12. forráskód. Egy csapat összes meccsének kigyűjtése

```
static List<meccs> meccsek(string csnev, List<meccs> lista)
{
   List<meccs> ret = new List<meccs>();
   foreach (meccs p in lista)
      if (p.csapat_1 == csnev || p.csapat_2 == csnev)
         ret.Add(p);
   //
   return ret;
}
```

17.13. forráskód. A zeneszámok rendezése csökkenően

```
static void rendezes(List<zeneszam> lista)
{
   for (int i = 0; i < lista.Count; i++)
      for (int j = i + 1; j < lista.Count; j++)
         if (lista[i].nepszeruseg< lista[j].nepszeruseg)
         {
            zeneszam c = lista[i];
            lista[i] = lista[j];
            lista[j] = c;
         }
}
```

17.14. forráskód. Az órarendi rekord

```
class orarend
{
   public string osztaly;
   public string tanar;
   public string tanterem;
   public string nap;
   public int ora_kezd;
   public int ora_hossza;
}
```

17.15. forráskód. Két órarendi bejegyzés ütközik-e (tanár szempontjából)

```
static bool utkozes_tanar(orarend a, orarend b)
{
 if (a == b)
      return false;
 if (a.tanar != b.tanar)
      return false;
 if (a.nap != b.nap)
      return false;
 if (a.ora_kezd<=b.ora_kezd && b.ora_kezd<a.ora_kezd+a.ora_hossza)
```

```
      return true;
 if (b.ora_kezd<=a.ora_kezd && a.ora_kezd<b.ora_kezd+b.ora_hossza)
      return true;
   //
   return false;
}
```

17.16. forráskód. Van-e órarendi ütközés (tanár szempontjából)

```
static bool utkozes_tanar(List<orarend> lista)
{
   foreach (orarend p in lista)
      foreach (orarend q in lista)
         if (utkozes_tanar(p, q))
            return true;
   //
   return false;
}
```

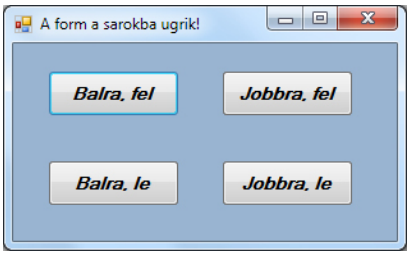
18. fejezet - Windows Form (szerző: Radványi Tibor)

Tartalom

A form és tulajdonságai

18.1. feladat (Form sarokba igazítása gombnyomásra – *szint: 1*). Készíts WindowsForm alkalmazást, mely a képernyő 4 sarkába ugorhat. Legyen rajta 4 *button*, amelyre kattintva a form – a feliratának megfelelő – sarokba ugrik.

18.1. ábra. Működés közben.



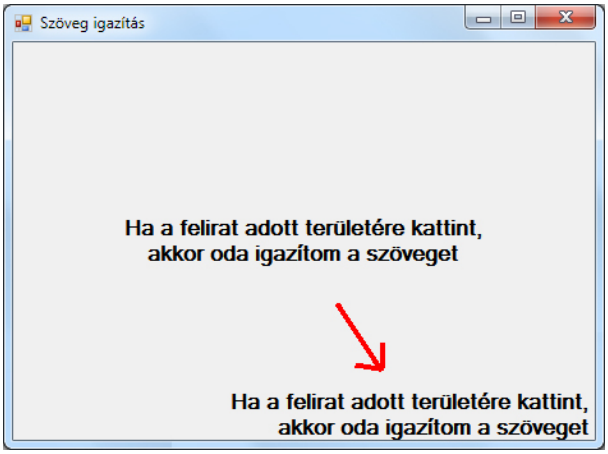
Magyarázat: Figyeljünk arra, hogy ne adjuk meg előre a lehetséges felbontás értékeit, helyette használjuk a *Screen.PrimeryScreen* objektum *Bounds* tulajdonságait.

18.1. forráskód. A Form sarokba igazítása

```
public partial class Frm_Sarokba : Form
{
      private void Bt_BF_Click(object sender, EventArgs e)
      {
            Left = 0;
            Top = 0;
      }

      private void Bt_JF_Click(object sender, EventArgs e)
      {
            Left = Screen.PrimaryScreen.Bounds.Width - Width;
            Top = 0;
      }

      private void Bt_BL_Click(object sender, EventArgs e)
      {
            Left = 0;
            Top = Screen.PrimaryScreen.Bounds.Height - Height;
      }

      private void Bt_JL_Click(object sender, EventArgs e)
      {
            Left = Screen.PrimaryScreen.Bounds.Width - Width;
            Top = Screen.PrimaryScreen.Bounds.Height - Height;
      }
}
```

18.2. feladat (Szöveg igazítása – *szint: 1*). Készíts WindowsForm alkalmazást, amely a *label* komponens szövegigazításait demonstrálja. A formon egy *label* legyen található, ami a form egész területét elfoglalja. A *label* egyes részeire kattintva változtathatjuk a szöveg igazítását.

18.2. ábra. Működés közben.



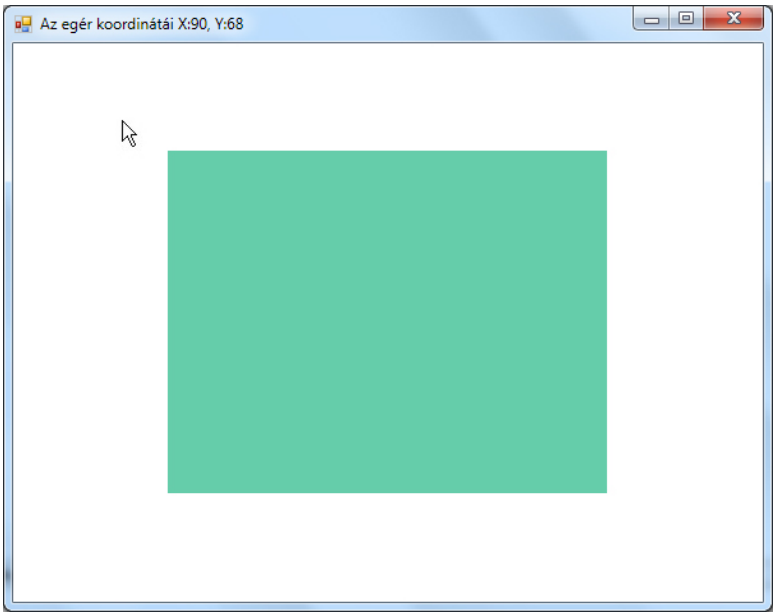
Magyarázat: A 9 külön lehetőség miatt ne hozzunk létre komponenseket, és írjunk külön eseménykezelőket. Helyette képzeletben osszuk fel a *form* területét, és a kattintásokkor ez alapján állítsuk be a *label* komponens szövegének elrendezését.

18.2. forráskód. A szöveg igazítása

```
private void label_MouseClick(object sender, MouseEventArgs e)
{
   int n = e.X / (label.Width / 3);
   int m = e.Y / (label.Height / 3);
   switch (m * 3 + n)
   {
      case 0: label.TextAlign = ContentAlignment.TopLeft; break;
      case 1: label.TextAlign = ContentAlignment.TopCenter; break;
      case 2: label.TextAlign = ContentAlignment.TopRight; break;
      case 3: label.TextAlign = ContentAlignment.MiddleLeft; break;
      case 4: label.TextAlign = ContentAlignment.MiddleCenter; break;
      case 5: label.TextAlign = ContentAlignment.MiddleRight; break;
      case 6: label.TextAlign = ContentAlignment.BottomLeft; break;
      case 7: label.TextAlign = ContentAlignment.BottomCenter; break;
      case 8: label.TextAlign = ContentAlignment.BottomRight; break;
   }
}
```

18.3. feladat (Form és Label koordinátáinak kiírása – *szint: 1*). Írjon WindowsForm alkalmazást, amely az egér aktuális pozícióját írja ki. A pozíció rögtön frissüljön a fejlécben, ahogy az egér megmozdul. Legyen a form közepén egy panel, ami fölé érve az egér pozíciója a panelhez relatív legyen, vagyis a panel bal felső sarkában 0,0.

18.3. ábra. Működés közben.



Magyarázat: Figyeljünk arra, hogy a panel a *form* közepén mindig pontosan középen legyen, és a *form* méretének negyedét foglalja csak el, akkor is, ha átméretezik az ablakot, ehhez iratkozzunk fel a form átméretezésének eseményére.

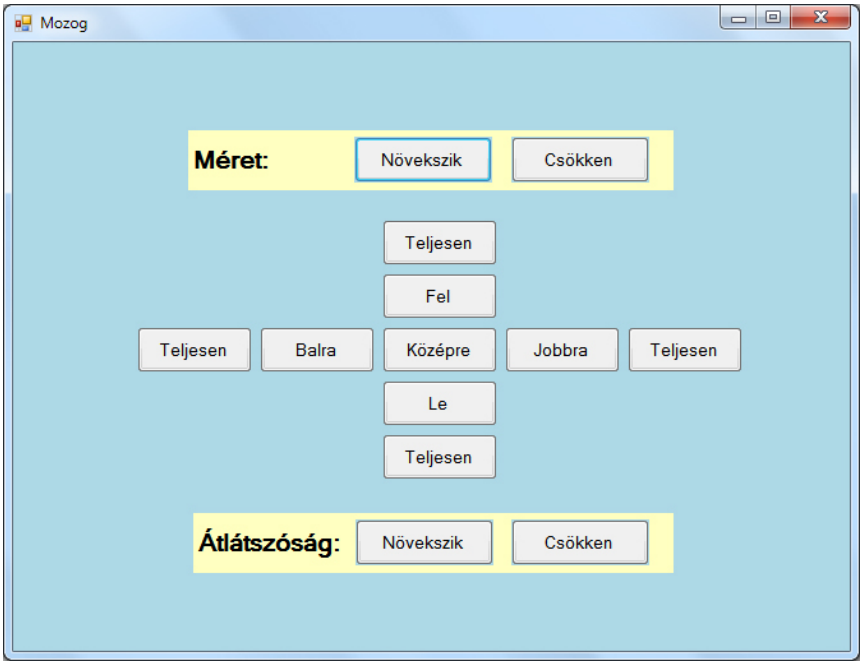
18.3. forráskód. Egér koordinátáinak kiírása

```
private void Frm_Holmozog_MouseMove(object sender, MouseEventArgs e)
{
   Text = String.Format("Az egér koordinátái X:{0}, Y:{1}", e.X, e.Y);
}

private void Frm_Holmozog_Resize(object sender, EventArgs e)
{
   PanelIgazit();
```

```
}

private void Frm_Holmozog_Load(object sender, EventArgs e)
{
   PanelIgazit();
}

private void PanelIgazit()
{
   panel.Left = (ClientSize.Width - panel.Width) / 2;
   panel.Top = (ClientSize.Height - panel.Height) / 2;
}
```

18.4. feladat (Form mozgatása – *szint: 2*). Írjon WindowsForm alkalmazást, amellyel a form pozíciójának, méretének és átlátszóságának változásait mutathatja be. Két gomb szolgáljon a form méretének csökkentésére és növelésére. Adjunk meg minimális méretet a formnak, ami alá nem csökkenhet. Az átlátszóság 20% alá ne mehessen. A *teljesen gombok*-ra ugorjon a form teljesen a képernyő szélére a megfelelő irányba, majd tűnjön el, hogy ne lehessen megnyomni ha már az ablak szélére került a form, viszont jelenjen meg újra, ha elhagytuk azt valamelyik másik gomb segítségével. A balra, fel, jobbra, le gombok konstansként meghatározott mértékben mozgassák az ablakot, és ugyancsak ne látszódjanak, ha már a képernyő szélére került a form. A *középre gomb* hatására a form foglalja el pontosan a képernyő közepét (ilyenkor látszódjon minden gomb).

18.4. ábra. Működés közben.



Magyarázat: Akárcsak az első feladatnál, itt is használjuk a *Screen.PrimeryScreen.Bounds* tulajdonságot a képernyő méreteihez. Figyeljünk arra, hogy ha elértük a határokat tüntessük el a megfelelő gombokat.

18.4. forráskód. Form jellemzői

```
private void Bt_M_Cso_Click(object sender, EventArgs e)
{
   Width -= meretezo;
   Height -= meretezo;
}

private void Bt_Atl_No_Click(object sender, EventArgs e)
{
   if (Opacity < 1.0)
      Opacity += 0.1;
}

private void Bt_Kozep_Click(object sender, EventArgs e)
{
   Left = (Screen.PrimaryScreen.Bounds.Width - Width) / 2;
   Top = (Screen.PrimaryScreen.Bounds.Height - Height) / 2;
   Bt_Le.Visible = Bt_Le_T.Visible =
   Bt_Fel.Visible = Bt_Fel_T.Visible =
   Bt_Bal.Visible = Bt_Bal_T.Visible =
   Bt_Jobb.Visible = Bt_Jobb_T.Visible = true;
}
```

18.5. forráskód. Form mozgatása

```
private void Bt_Fel_T_Click(object sender, EventArgs e)
{
   Top = 0;
   Bt_Fel.Visible = Bt_Fel_T.Visible = false;
   Bt_Le.Visible = Bt_Le_T.Visible = true;
}
```

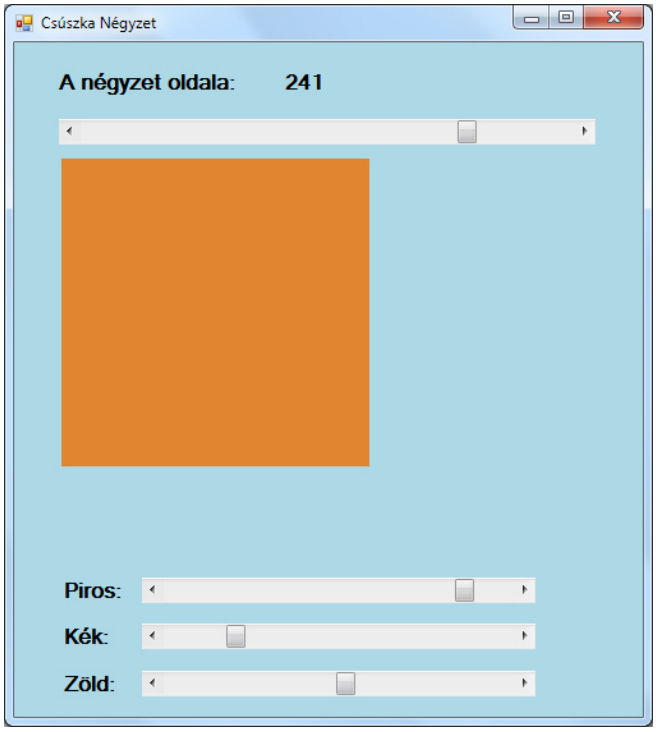
```
private void Bt_Fel_Click(object sender, EventArgs e)
{
   if (Top - meretezo > 0)
   {
      Top -= meretezo;
   }
   else
   {
      Top = 0;
      Bt_Fel.Visible = Bt_Fel_T.Visible = false;
   }
   Bt_Le.Visible = Bt_Le_T.Visible = true;
}
```

Alapvető komponensek, adatbekérés és megjelenítés

18.5. feladat (ScrollBar használata – *szint: 2*). Írjon programot, amely az additív színkeverést mutatja be. Egy négyzet alakú label háttérszíne mutassa a kevert színt. Három scrollbar komponenssel lehessen állítani a szín piros, kék, és zöld összetevőjét. Akármelyik csúszka pozíciója változik, rögtön frissüljön a szín. A színt mutató label méretét szintén egy csúszkával lehessen változtatni, ez 10 és 300 pixel közötti érték lehet.

Magyarázat: Ne felejtse el beállítani a scrollbarok minimum és maximum értékeit.

18.5. ábra. Működés közben.



18.6. forráskód. ScrollBar használata

```
public partial class Frm_Csuszka : Form
{
   private void Sb_Piros_ValueChanged(object sender, EventArgs e)
   {
      Lb_Negyzet.BackColor = Color.FromArgb(Sb_Piros.Value,
                       SB_Zold.Value, SB_Kek.Value);
   }

   private void Sb_Oldal_ValueChanged(object sender, EventArgs e)
   {
      Lb_Negyzet.Width = Lb_Negyzet.Height = Sb_Oldal.Value;
      Lb_Oldal.Text = Sb_Oldal.Value.ToString();
   }
}
```

18.6. feladat (Képet forgat – *szint: 1*). Írjon programot, mely köralakban jelenít meg képeket, és mindkét irányba körbe léptethetőek.

18.6. ábra. Működés közben.

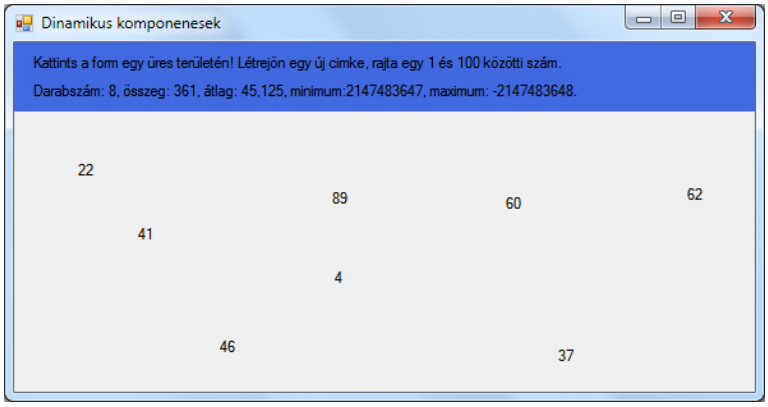
Magyarázat: A képek egymás közti forgatásához vegyünk fel egy Image típusú segédváltozót.

18.7. forráskód. Forgat

```
public partial class Frm_Kepforog : Form
{
   private void Pb_Bal_Click(object sender, EventArgs e)
   {
      Image s = Lb_1.Image;
      Lb_1.Image = Lb_2.Image;
      Lb_2.Image = Lb_3.Image;
      Lb_3.Image = Lb_4.Image;
      Lb_4.Image = Lb_5.Image;
      Lb_5.Image = Lb_6.Image;
      Lb_6.Image = Lb_7.Image;
      Lb_7.Image = Lb_8.Image;
      Lb_8.Image = s;
   }
}
```

18.7. feladat (Futásidejű komponensek – *szint: 2*). Készítsen programot, ami minden egérkattintáskor egy-egy új label komponenst hoz létre dinamikusan. Az új label felirata legyen egy véletlen szám 1 és 100 között. Számoljuk, hogy mennyi új komponens jött létre, mi a véletlen számok összege, átlaga, minimuma és maximuma.

18.7. ábra. Működés közben.



Magyarázat: A *min* és *max* változók inicializálásánál célszerű az int típus szélsőértékeit használni, hiszen pl. a int.MinValue-nál csak nagyobb számot generálhatunk. Véletlen szám generálásnál figyeljünk a határokra, a Random(100) 0 és 99 közötti számot generál, ezt még el kell tolni egyel. Az újonnan létrehozott *label* komponenst ne felejtsük el hozzáadni a *form Controls* tömbjéhez, hisz csak ekkor fog megjelenni. Átlagszámításnál figyeljünk arra, hogy az osztót (db) lebegőpontosra kell alakítani, mert különben egész osztást alkalmaz a fordító.

18.8. forráskód. Futás idejű létrehozás

```
public partial class Frm_Dinamikus : Form
{Random random = new Random();
double atlag; int osszeg, db, min = int.MaxValue, max = int.MinValue;}
private void Frm_Dinamikus_MouseClick(object sender, MouseEventArgs e)
{  int i = random.Next(100) + 1;
   Label lb = new Label();
   lb.Location = new Point(e.X, e.Y);
   lb.Text = i.ToString();
```

```
   lb.AutoSize = true;
   Controls.Add(lb);
   db++;
   osszeg += i;
   atlag = osszeg / (double)db;
   if (min > i) min = i;
   if (max < i) max = i;
   lb_Eredmeny.Text =
    String.Format("Darabszám: {0}, összeg: {1}, átlag: {2},"
              +"minimum: {3}, maximum: {4}.",
              db, osszeg, atlag, min, max); } }
```

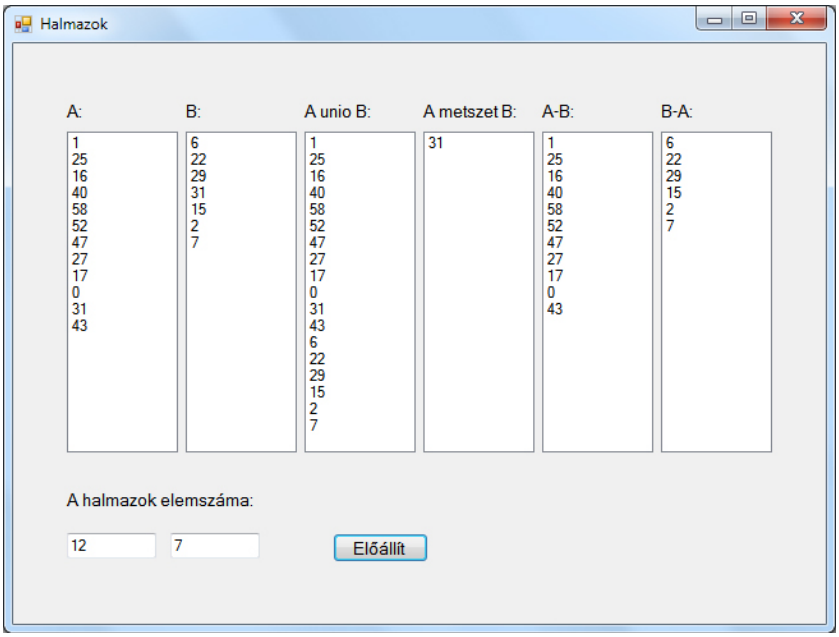
18.8. feladat (Halmazok – *szint: 3*). Írjon programot, amely bemutatja két halmaz közötti műveleteket. Egy-egy halmaznak egy-egy listbox komponensek feleljen meg. Kezdetben legyen üres mind az A, és B halmaz is. Mindkét halmaz elemszámát kérjük be, majd töltsük fel őket véletlen számokkal, és számoljuk ki A unió B -t, A metszet B-t, A-B és B-A eredményét is.

Magyarázat: A feltöltésnél figyeljünk arra, hogy nem csak egyszerű adatsorozatot készítünk, hanem halmazt, melynek fő tulajdonsága, hogy egy elem csak egyszer szerepelhet benne.

18.9. forráskód. Halmaz generálása

```
private void Bt_Eloall_Click(object sender, EventArgs e)
{
   HalmazGeneral(LBx_A, Convert.ToInt32(TBx_A.Text));
   HalmazGeneral(LBx_B, Convert.ToInt32(TBx_B.Text));
   Metszet(LBx_A, LBx_B, LBx_Metsz);
   Unio(LBx_A, LBx_B, LBx_Unio);
   Minusz(LBx_A, LBx_B, LBx_A_B);
   Minusz(LBx_B, LBx_A, LBx_B_A);
}

private void HalmazGeneral(ListBox LB, int N)
{
   int elem;
   LB.Items.Clear();
   for (int i = 0; i < N; i++)
   {
      do
      {
       elem = random.Next(N * 5);
      }
      while (Bennevan(LB, elem));
      LB.Items.Add(elem);
   }
}

private bool Bennevan(ListBox LB, object elem)
{
   for (int i = 0; i < LB.Items.Count; i++)
      if (LB.Items[i].Equals(elem))
            return true;
   return false;
}
```

18.8. ábra. Működés közben.



18.10. forráskód. Halmazműveletek

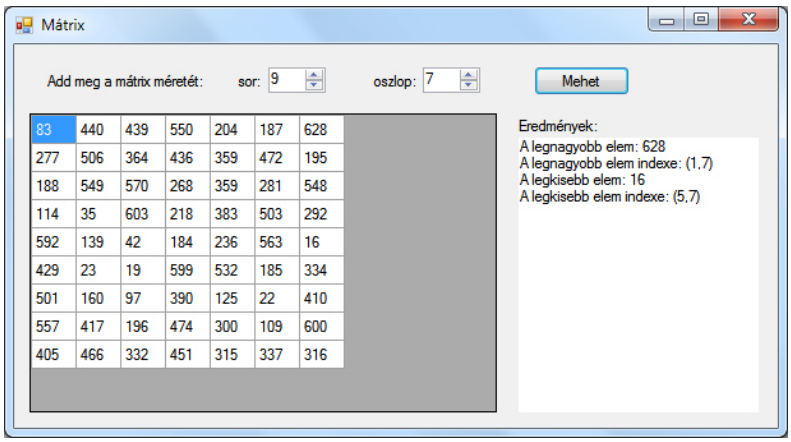
```
private void Unio(ListBox LBx_A, ListBox LBx_B, ListBox LBx_Unio)
{
   LBx_Unio.Items.Clear();
   for (int i = 0; i < LBx_A.Items.Count; i++)
   {
      if (!Bennevan(LBx_Unio, LBx_A.Items[i]))
      {
```

```
            LBx_Unio.Items.Add(LBx_A.Items[i]);
         }
      }
      for (int i = 0; i < LBx_B.Items.Count; i++)
      {
         if (!Bennevan(LBx_Unio, LBx_B.Items[i]))
         {
            LBx_Unio.Items.Add(LBx_B.Items[i]);
         }
      }
   }

   private void Metszet(ListBox LBx_A, ListBox LBx_B, ListBox LBx_Metsz)
   {
      LBx_Metsz.Items.Clear();
      for (int i = 0; i < LBx_A.Items.Count; i++)
      {
         if (Bennevan(LBx_B, LBx_A.Items[i]))
         {
            LBx_Metsz.Items.Add(LBx_A.Items[i]);
         }
      }
   }
```

18.9. feladat (Mátrix – *szint: 4*). Készítsen programot, mely egy mátrix elemeit feltölti véletlen számokkal, majd kiírja melyik a legkisebb, ill. legnagyobb szám, és hol vannak. A mátrixot jelenítse meg egy *DataGridView* komponensben, oszlopainak és sorainak száma 2 és 10 közötti - nem feltétlenül egyenlő - számok legyenek, és a felhasználó adhassa meg *NumericUpDown* komponenssel.

Magyarázat: A *NumericUpDown* komponensnél ne felejtsük el beállítani a határokat. A *grid* kialakításánál figyeljünk oda, hogy letiltsuk azt *(ReadOnly property)*, illetve ne látszódjanak az oszlop- és sorfejlécek *(ColumnHeadersVisible, RowHeadersVisible property-k)*.

18.9. ábra. Működés közben.



Magyarázat: Feltöltéskor ne felejtsük el létrehozni az oszlopokat a táblázatba, és csak ezután adjuk hozzá a véletlen számokból álló sorokat.

18.11. forráskód. Feltölt

```
private void AdatokFeltolt(int N, int M)
{
   dataGridView.Columns.Clear();
   dataGridView.Rows.Clear();

   for (int j = 0; j < M; j++)
   {
      dataGridView.Columns.Add(String.Empty, String.Empty);
      dataGridView.Columns[j].Width = 35;
   }
   for (int i = 0; i < N; i++)
   {
      object[] intArrray = new object[M];
      for (int j = 0; j < M; j++)
      {
         intArrray[j] = random.Next(N * M * 10) + 1;
      }
      dataGridView.Rows.Add(intArrray);
   }
}
```

Magyarázat: A táblázat feldolgozásakor ne felejtsük, hogy a táblázat cellái object típusúként vannak eltárolva, felhasználásukkor át kell alakítani (castolni) int típusúra őket.

A eredmény kiárásához használt *RichTextBox* komponens szövegét ne felejtsük el alaphelyzetbe állítani *(ResetText metódus)*, és csak ezután írjunk bele *(AppendText)*.

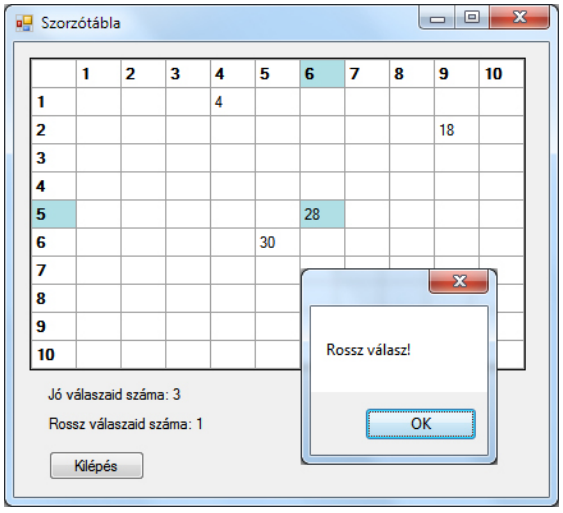
18.12. forráskód. Eredmény

```
private void EredmenyKiir(int N, int M)
{
   int max = int.MinValue,
      min = int.MaxValue;

   int maxi, maxj, mini, minj;
   maxi = maxj = mini = minj = 0;

   for (int i = 0; i < N; i++)
```

```
   {
      for (int j = 0; j < M; j++)
      {
         if ((int)dataGridView[j, i].Value > max)
         {
            max = (int)dataGridView[j, i].Value;
            maxi = i;
            maxj = j;
         }
         if ((int)dataGridView[j, i].Value < min)
         {
            min = (int)dataGridView[j, i].Value;
            mini = i;
            minj = j;
         }
      }
   }
   richTextBox.ResetText();

   richTextBox.AppendText(String.Format("A legnagyobb elem: {0}{1}",
            max, Environment.NewLine));
   richTextBox.AppendText(String.Format("A legnagyobb elem indexe:"
        +"({0},{1}){2}", maxi + 1, maxj + 1, Environment.NewLine));
   richTextBox.AppendText(String.Format("A legkisebb elem: {0}{1}",
            min, Environment.NewLine));
   richTextBox.AppendText(String.Format("A legkisebb elem indexe:"
        +"({0},{1}){2}", mini + 1, minj + 1, Environment.NewLine));
}
```

18.10. feladat (Szorzótábla – *szint: 4*). Készítsen programot a szorzótábla gyakorlására. A program véletlenszerűen kérdezzen rá a szorzatokra, jelezze az első sorban és oszlopban más háttérszínnel, hogy éppen melyik két szám szorzatára vagyunk kíváncsiak. Ugyancsak legyen más színű az a cella, ahová az eredményt várjuk, és rajta kívül egyetlen másik cella se legyen szerkeszthető. Elhagyva a cellát a program értékelje a számításunkat, ha jó számot adunk meg, akkor kérje a következő szorzatot, ha rosszat, akkor kérje be újra. A felhasználó két *label*-ben lássa, hogy hány jó, ill. rossz választ adott meg eddig. Ha minden cellát helyesen kitöltött egy üzenetablakban gratuláljunk neki.

18.10. ábra. Működés közben.



Magyarázat: A 9. feladathoz hasonlóan itt is tiltsuk le a *gridben* a sor és oszlop fejléceket, de ne állítsuk csak szerkeszthetőre *(readonly)*. A *grid* szerkesztési módját *(EditMode)* állítsuk *EditOnEnter*-re.

18.13. forráskód. Sorsol egy pozíciót

```
private void Sorsol()
{
   do
   {
      aktI = random.Next(N) + 1;
      aktJ = random.Next(N) + 1;
   } while (dataGridView[aktJ, aktI].Value != null);
   dataGridView[aktJ, aktI].Style.BackColor =
      dataGridView[0, aktI].Style.BackColor =
      dataGridView[aktJ, 0].Style.BackColor = Color.PowderBlue;
}
```

18.14. forráskód. Feltölti a fejléceket

```
private void TablaEpit()
{
   dataGridView.Columns.Clear();
   dataGridView.Rows.Clear();
   dataGridView.Width = 35 * (N + 1) + 3;
   for (int j = 0; j <= N; j++)
   {  dataGridView.Columns.Add(String.Empty, String.Empty);
      dataGridView.Columns[j].Width = 35; }
   for (int i = 0; i <= N; i++)
   {  object[] intArrray = new object[N];
      dataGridView.Rows.Add(intArrray); }
   for (int i = 1; i <= N; i++)
   {  dataGridView[i, 0].Value = i;
      dataGridView[i, 0].Style.Font = new Font(dataGridView.Font,
```

```
                                    FontStyle.Bold); }

   for (int j = 1; j <= N; j++)
   {  dataGridView[0, j].Value = j;
      dataGridView[0, j].Style.Font = new Font(dataGridView.Font,
                                    FontStyle.Bold); }
}
```

Magyarázat: Az új kérdés sorsolásakor figyeljünk arra, hogy olyan szorzatot válasszunk, amit még nem töltött ki a felhasználó, és sor-, ill. oszlopindexét tároljuk el változókban, ezek alapján tudjuk figyelni a *grid CellBeginEdit* eseményében, hogy a megfelelő cellát kezdi-e el szerkeszteni a felhasználó.

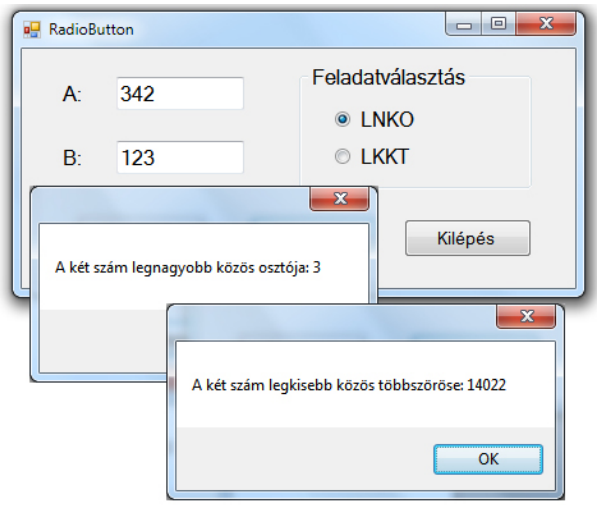
18.15. forráskód. Ellenőriz

```
private void dataGridView_CellEndEdit(object sender,
                 DataGridViewCellEventArgs e)
{
   if (dataGridView[aktJ, aktI].Value == null) return;

   if (aktI * aktJ == Convert.ToInt32(dataGridView[aktJ, aktI].Value))
   {  dataGridView[aktJ, aktI].Style.BackColor =
            dataGridView.DefaultCellStyle.BackColor;
      dataGridView[0, aktI].Style.BackColor =
         dataGridView[aktJ, 0].Style.BackColor =
         dataGridView.DefaultCellStyle.BackColor;
      joValasz++;
      lb_Joalasz.Text = String.Format("Jó válaszaid száma: {0}",
                           joValasz);
      if (joValasz < N * N)
      { Sorsol(); }
      else
      { MessageBox.Show("Gratulálok, kész a szorzótábla!");
       dataGridView.Enabled = false; }
   }
   else
   {  dataGridView[aktJ, aktI].Value = null;
      rosszValasz++;
      lb_Rosszvalasz.Text = String.Format("Rossz válaszaid száma:"
                              +"{0}", rosszValasz);
      MessageBox.Show("Rossz válasz!"); }
}
```

Választások

18.11. feladat (LNKO LKKT – *szint: 1*). Írjon programot, mely bekér két egész számot, és kiszámolja azok legnagyobb közös osztóját (LNKO) és legkisebb közös többszörösét (LKKT). Hogy épp melyiket számolja ki, azt rádiógombokkal lehessen beállítani. Külön gomb szolgáljon a kilépésre.

18.11. ábra. Működés közben.



Magyarázat: Elég megírjuk az lnko algoritmusát, hiszen a legkisebb közös többszöröst az (a*b)/LNKO(a,b) képlet megadja.

18.16. forráskód. Kiszámolás

```
public int LNKO(int a, int b)
{
   if (a == 0)
      return b;
   if (b == 0)
      return a;

   if (a > b)
      return LNKO(a % b, b);
   else
      return LNKO(a, b % a);
}
```

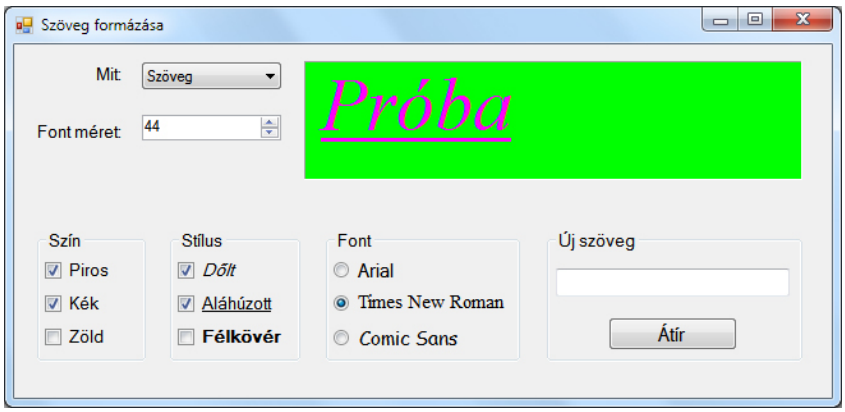
Magyarázat: Az *int.TryParse(...)* metódusával ellenőrizzük le hogy tényleg számot adott-e meg a felhasználó, és csak ez után számoljunk.

18.17. forráskód. Művelet elvégzése

```
private void button1_Click(object sender, EventArgs e)
{
   int a, b;

   // Adatok ellenőrzése számítás előtt

   if (!int.TryParse(TBx_A.Text, out a))
   {
      MessageBox.Show("Nem megfelelő szám!");
      TBx_A.Text = String.Empty;
      TBx_A.Focus();
      return;
   }

   if (!int.TryParse(TBx_B.Text, out b))
   {
      MessageBox.Show("Nem megfelelő szám!");
      TBx_B.Text = String.Empty;
      TBx_B.Focus();
      return;
   }

   if (Rb_LNKO.Checked)
   {
      MessageBox.Show(String.Format(
         "A két szám legnagyobb közös osztója: {0}",
         LNKO(a, b)));
   }
   else
   {
      MessageBox.Show(String.Format(
         "A két szám legkisebb közös többszöröse: {0}",
         (a * b) / LNKO(a, b)));
   }
}
```

18.12. feladat (Formázás – *szint: 2*). Készítsünk alkalmazást, mely egy szöveg betűtípusának beállításait mutatja be. Lehessen beállítani a betűtípus méretét, stílusát (dőlt, aláhúzott, félkövér), típusát (3 választás: Arial, Times New Roman, Comic Sans). Használjunk *ComboBox* és *CheckBox* komponenseket. Állítható legyen továbbá a *label* szövegének és hátterének színe is. Adjon lehetőséget a program a *label* szövegének átírására is.

18.12. ábra. Működés közben.



18.18. forráskód. Változtatás kezelése

```
private void Cbx_Piros_CheckedChanged(object sender, EventArgs e)
{
  if (Comb_Mit.SelectedIndex == 0)
  {
     Lb_Proba.ForeColor = SzinBeallit();
  }
  else
  {
     Lb_Proba.BackColor = SzinBeallit();
  }
}
```

18.19. forráskód. Font beállításai

```
private void FontBeallit()
{
   Lb_Proba.Font = new Font(GetFontName(),
      (int)NDD_Fontmeret.Value, GetFontStyle());
}

private FontStyle GetFontStyle()
{
   FontStyle result = FontStyle.Regular;
   if (Cbx_Alahuz.Checked)
      result |= FontStyle.Underline;
   if (Cbx_Dolt.Checked)
      result |= FontStyle.Italic;
   if (Cbx_Felk.Checked)
      result |= FontStyle.Bold;
   return result;
}
```

```
private string GetFontName()
{
   if (Rb_Arial.Checked)
      return Rb_Arial.Text;
   else if (Rb_CS.Checked)
      return Rb_CS.Text;
   else if (Rb_TNR.Checked)
      return Rb_TNR.Text;
   return String.Empty;
}
```

Magyarázat: A *GetFontStyle* metódusban használjuk ki, hogy a *FontStyle* felsorolástípus maszkolható, vagyis az aláhúzás, dőlt betű és a vastag betűs tulajdonságokat egymástól függetlenül is be lehet állítani a '|' operátor segítségével.

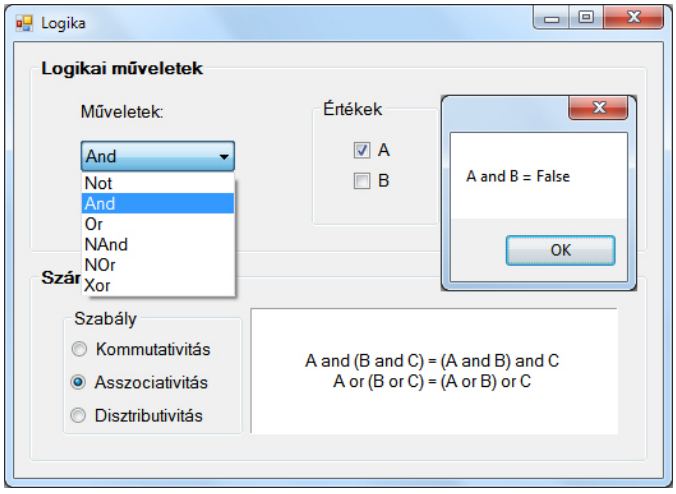
18.20. forráskód. Szín beállítás és ComboBox

```
private Color SzinBeallit()
{
   return Color.FromArgb(
      Cbx_Piros.Checked ? 255 : 0,
      Cbx_Zold.Checked ? 255 : 0,
      Cbx_Kek.Checked ? 255 : 0
   );
}

private void Comb_Mit_SelectedIndexChanged(object sender, EventArgs e)
{
   GB_Font.Enabled = GB_Stilus.Enabled = NDD_Fontmeret.Enabled =
      (Comb_Mit.SelectedIndex == 0);
   CheckBoxokBeallit((Comb_Mit.SelectedIndex == 0) ?
      Lb_Proba.ForeColor : Lb_Proba.BackColor);
}
```

Magyarázat: A színek beállításánál a kijelöléstől függően állítsuk a szín adott komponensét 0, vagy 255 értékre.

18.13. feladat (Logikai műveletek – *szint: 2*). Készítsen programot, mely bemutatja a matematikai logikai műveleteket. Egy combobox-ból lehessen kiválasztani az aktuális műveletet, az operandusok (A és B, vagy csak A) értékét checkbox komponenssel állítsuk be. Ezen kívül a program írja ki a számolási szabályokat a műveletek között: kommutativitás, asszociativitás, disztributivitás.

18.13. ábra. Működés közben.



Magyarázat: Figyeljünk arra, hogy attól függően, hogy 1 vagy 2 operandusú az operátor, csak annyi bemeneti adat *(CheckBox)* legyen kötelező.

18.21. forráskód. ComboBox kezelése

```
private void Frm_Logika_Load(object sender, EventArgs e)
{
   comboBoxMuvelet.SelectedIndex = 0;
}

private void comboBoxMuvelet_SelectedIndexChanged(object sender,
      EventArgs e)
{
   Cbx_B.Enabled = comboBoxMuvelet.SelectedIndex != 0;
}
```

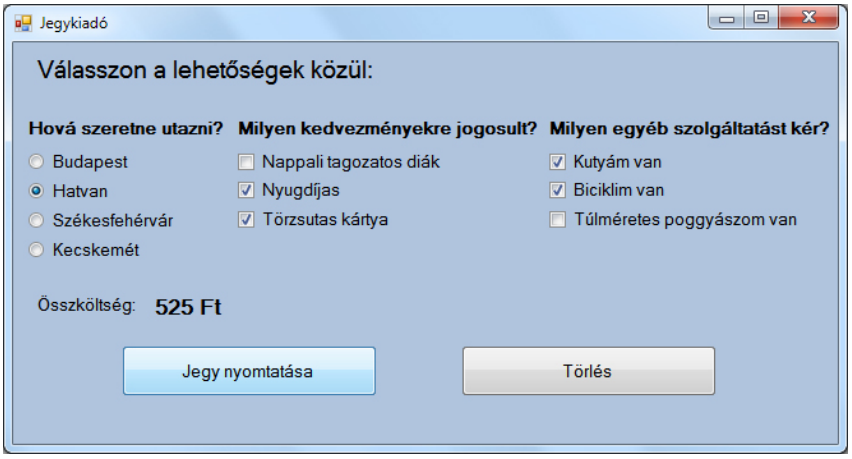
18.22. forráskód. Logikai műveletek

```
private void Bt_Eredmeny_Click(object sender, EventArgs e)
{
   switch (comboBoxMuvelet.SelectedIndex)
   {
   case 0: MessageBox.Show(String.Format("not A = {0}",
                           !Cbx_A.Checked));
         break;
   case 1: MessageBox.Show(String.Format("A and B = {0}",
                     Cbx_A.Checked & Cbx_B.Checked));
         break;
   case 2: MessageBox.Show(String.Format("A or B = {0}",
```

```
                    Cbx_A.Checked | Cbx_B.Checked));
            break;
        case 3: MessageBox.Show(String.Format("A nand B = {0}",
                    !(Cbx_A.Checked & Cbx_B.Checked)));
            break;
        case 4: MessageBox.Show(String.Format("A nor B = {0}",
                    !(Cbx_A.Checked | Cbx_B.Checked)));
            break;
        case 5: MessageBox.Show(String.Format("A xor B = {0}",
                    Cbx_A.Checked ^ Cbx_B.Checked));
            break;
    }
}


private void Rb_Komm_CheckedChanged(object sender, EventArgs e)
{
    if (Rb_Komm.Checked)
    {
        Lb_SzSz.Text = "A and B = B and A" + Environment.NewLine +
                    "A or B = B or A";
    }
    else if (Rb_Assz.Checked)
    {
        Lb_SzSz.Text = "A and (B and C) = (A and B) and C" +
                    Environment.NewLine +
                    "A or (B or C) = (A or B) or C";
    }
    else if (Rb_Disz.Checked)
    {
        Lb_SzSz.Text = "A and (B or C) = (A and B) or (A and C)" +
                    Environment.NewLine +
                    "A or (B and C) = (A or B) and (A or C)";
    }
}
```

18.14. feladat (Jegykiadó program – *szint: 2*). Írjon programot, ami különböző paraméterek alapján kiszámol egy vonatjegy árat. Lehessen megadni, hogy hova utazunk (Budapest 1500Ft, Hatvan 1300Ft, Székesfehérvár 2000Ft, Kecskemét 2800Ft), milyen kedvezményünk van (nappali tagozatos diák 50%, nyugdíjas 50%, törzsutas kártya 70%), és hogy milyen egyéb szolgáltatásokat kell igénybe vennünk (kutya 800Ft, bicikli 600Ft, túlméretes poggyász 400Ft). A beállítások után egy gombra kattintva írja ki a program a fizetendő árat, egy másik gomb pedig adjon lehetőséget a beállítások törlésére.

18.14. ábra. Működés közben.



Magyarázat: Nem lehet igénybe venni egyszerre nappali tagozatos diák és nyugdíjas kedvezményt.

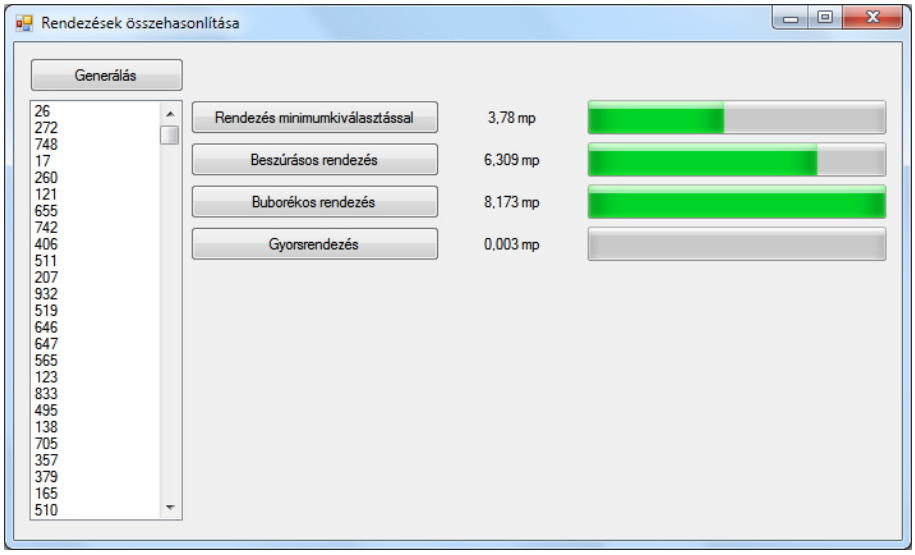
18.23. forráskód. Számol

```
private void NyomtatásButton_Click(object sender, EventArgs e)
{
    double díj = 0;

    if (Cbx_Napp.Checked && Cbx_Nyug.Checked)
    {  Lb_Koltseg.Text = "Hiba!";
       return;
    }
    if (Rb_Bud.Checked) díj = 1500;
    if (Rb_Hat.Checked) díj = 1300;
    if (Rb_Szek.Checked) díj = 2000;
    if (Rb_Kecs.Checked) díj = 2800;
    if (Cbx_Kutya.Checked) díj = díj + 800;
    if (Cbx_Bicikli.Checked) díj = díj + 600;
    if (Cbx_Poggy.Checked) díj = díj + 400;
    if (Cbx_Napp.Checked) díj = díj * 0.5;
    if (Cbx_Nyug.Checked) díj = díj * 0.5;
    if (Cbx_Torzs.Checked) díj = díj * 0.7;
    Lb_Koltseg.Text = díj + " Ft"; }
```

Listák kezelése

18.15. feladat (Rendezések hatékonysága – *szint: 3*). Készítsen programot, mely rendezéseket hasonlít össze. Generáljunk egy ListBox komponensbe 10000 és 20000 közti véletlen darabszámú 0 és 1000 közötti véletlenszámot. Ezeken az elemeken mérjük le a minimumkiválasztos, beszúrásos, buborékos rendezéseket, valamint a ListBox beépített rendezését *(.Sort())*. Írjuk ki az egyes rendezések után, hány másodpercbe kerültek. 1-1 *ProgressBar* komponensen érzékeltessük az arányokat. Ha új számokat generálunk, töröljük az eddigi időeredményeket.

18.15. ábra. Működés közben.



Magyarázat: Amíg nincs meg az adott rendezéshez tartozó időeredmény, addig a hozzá tartozó *label* és *progressbar* ne látszódjon a felületen.

18.24. forráskód. Véletlen számok generálása

```
private void Btn_General_Click(object sender, EventArgs e)
{
   Random rnd = new Random();
   Int32 db = rnd.Next(10000) + 10000, akt;
   listBox.Items.Clear();
   listBox.BeginUpdate();
   for (Int32 i = 0; i < db; i++)
   {
      akt = rnd.Next(1000);
      listBox.Items.Add(akt);
      list.Add(akt);
   }
   listBox.EndUpdate();
   MessageBox.Show(String.Format("Generált elemek száma: {0}", db));
   gombok.ForEach(bt => bt.Enabled = true);
   LB_Beszurasos.Visible = LB_Buborek.Visible =
       LB_Gyors.Visible = LB_Minkiv.Visible =
       PB_Beszurasos.Visible = PB_Buborek.Visible =
       PB_Gyorsrend.Visible = PB_Minkiv.Visible = false;
   eredmenyek.ForEach(pb => pb.Maximum = Int32.MaxValue);
}
```

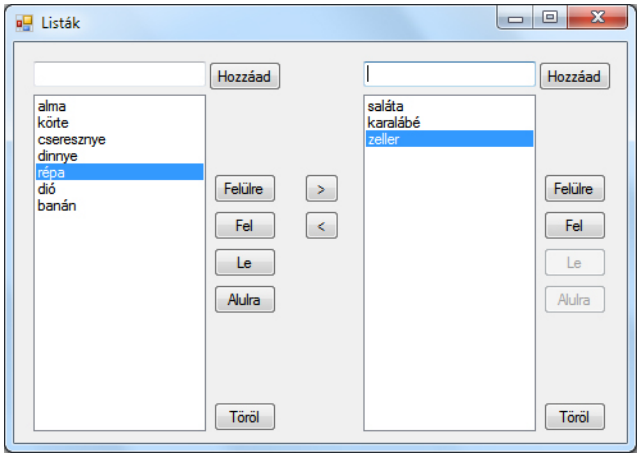
Magyarázat: A számok generálása közben érdemes kikapcsolni a *ListBox* komponens azonnali frissítését, hogy ne villogjon, és görgessen minden új elem hozzáadásakor. Ezt a *BeginUpdate* metódussal tehetjük meg, és a feltöltés után az *EndUpdate* hatására frissül a felületen a *listbox*. A gombok tömbben található *Button* objektumok mindegyikének engedélyezéséhez használjuk a generikus *List<>* adatszerkezet *ForEach* metódusát, ami egy *delegate*-et vár, és megadható neki *lambda-kifejezés* is (bt => bt.Enabled = true).

18.25. forráskód. Minimumkiválasztásos rendezés

```
private void Btn_Minkiv_Click(object sender, EventArgs e)
{
   gombok.ForEach(bt => bt.Enabled = false);
   DateTime start = DateTime.Now;
   MinKivRend(new List<Int32>(list));
   Double eredmeny = (DateTime.Now - start).TotalSeconds;
   LB_Minkiv.Text = String.Format("{0} mp", eredmeny);
   LB_Minkiv.Show();
   PB_Minkiv.Value = PB_Minkiv.Maximum =
      (Int32)Math.Round(eredmeny * 100);
   Maxok();
   PB_Minkiv.Show();
   gombok.ForEach(bt => bt.Enabled = true);
}
```

18.16. feladat (Lista karbantartása – *szint: 3*). Készítsünk programot, mely két *ListBox* karbantartását végzi el. Mindkét *ListBox*-on a következő műveleteket végezhetjük el: új elem felvétele, kijelölt elem törlése, kijelölt elem mozgatása (felülre, fel, le, alulra) Ehhez készítsünk külön komponenst, mely tartalmaz egy *ListBox*-ot, és a fenti műveletekhez 1-1 gombot. Egy *TextBox*-on kérje be az új elemet, és ez is legyen része a komponensnek. A főablakra két ilyen komponenst tegyünk fel, majd egészítsük ki még két publikus metódussal: RemoveSelectedItem, AddNewItem. Ezek segítségével oldjuk meg a két komponens között az elemek átadását.

18.16. ábra. Működés közben.



Magyarázat: A komponens készítésekor ügyeljünk arra, hogy a funkciókat letiltsuk, ha azoknak nincs értelme, pl. üres listából értelmetlen törölni, és a legfelső elemet sem lehet feljebb mozgatni.

18.26. forráskód. Komponensek közti mozgatás

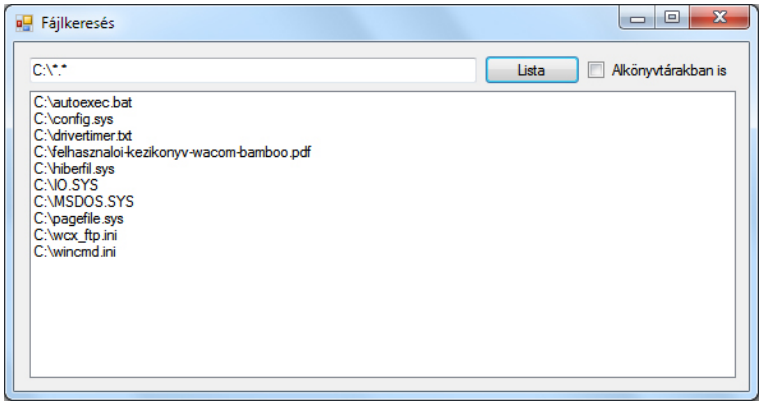
```
private void button3_Click(object sender, EventArgs e)
{
   Object item = listBoxControl1.RemoveSelectedItem();
   if (item != null)
   {
      listBoxControl2.AddNewItem(item);
   }
}

private void button4_Click(object sender, EventArgs e)
{
   Object item = listBoxControl2.RemoveSelectedItem();
   if (item != null)
   {
      listBoxControl1.AddNewItem(item);
   }
}
```

18.27. forráskód. Komponens főbb funkciói

```
private void Btn_Hozzaad_Click(object sender, EventArgs e)
{  if (TB_Uj.Text != String.Empty)
   {
      LB.Items.Add(TB_Uj.Text);
      TB_Uj.Text = String.Empty;
      LB.SelectedIndex = LB.Items.Count - 1;
      GombokFrissit();
   }
   TB_Uj.Focus();
}

private void Btn_Torles_Click(object sender, EventArgs e)
{  Int32 selIndex = LB.SelectedIndex;
   if (selIndex >= 0)
   {
      LB.Items.RemoveAt(selIndex);
      if (selIndex < LB.Items.Count)
         LB.SelectedIndex = selIndex;
      else
         LB.SelectedIndex = LB.Items.Count - 1;
      GombokFrissit();
   }
}

private void Btn_Fel_Click(object sender, EventArgs e)
{  Int32 selIndex = LB.SelectedIndex;
   if (selIndex > 0)
   {
      Object selItem = LB.SelectedItem;
      LB.Items.RemoveAt(selIndex);
      LB.Items.Insert(selIndex - 1, selItem);
      LB.SelectedIndex = selIndex - 1;
   }
}
```

18.17. feladat (Fájl keresése – *szint: 3*). Írjunk programot, amely megadott elérési úton keres egy maszknak megfelelő fájlokat. Egy *TextBox*-ban kérjük be az elérési utat és a maszkot, pl. `"C:\*.txt"`. Meg lehessen adni egy *CheckBox* komponens segítségével, hogy rekurzívan keressünk-e, vagy csak az adott könyvtárban. A talált fájlokat egy *ListBox* komponensben jelenítsük meg.

18.17. ábra. Működés közben.



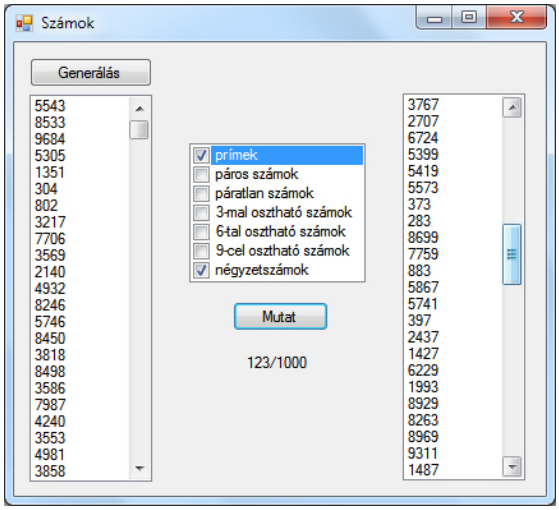
18.28. forráskód. Könyvtárszerkezet bejárása

```
public void GetFileList(String dir, String mask, Boolean recursive,
                 Action<String> action)
{DirectoryInfo di = new DirectoryInfo(dir);
   try {
      FileInfo[] rgFiles = di.GetFiles(mask);
      foreach (FileInfo fi in rgFiles)
            action(fi.FullName);
      if (recursive)
      { DirectoryInfo[] dirs = di.GetDirectories();
         foreach (DirectoryInfo dirInfo in dirs)
          GetFileList(Path.Combine(dir, dirInfo.Name),
                      mask, recursive, action); } }
   catch (UnauthorizedAccessException)
   {   }
}
private void Btn_Lista_Click(object sender, EventArgs e)
{  String dir = Path.GetDirectoryName(textBox.Text);
   String mask = textBox.Text.Substring(dir.Length);
   listBox.Items.Clear(); listBox.BeginUpdate();
   GetFileList(dir, mask, checkBox1.Checked,
           name => listBox.Items.Add(name));
   listBox.EndUpdate(); }
```

Magyarázat: A bejárás paramétere egy *Action<String>* típusú *delegate*, ezt a fájlnevet váró akciót fogja végrehajtani a bejárás minden találatra. A bejárás során kivételt kaphatunk, ha olyan könyvtárat akarunk megnyitni, amihez nincs jogunk, készüljünk fel erre is.

18.18. feladat (Számok kiválogatása – *szint: 3*). Írjunk programot, amely egy *ListBox*-ba generál véletlen számokat és egy *CheckListBox*-ban megadott feltételek alapján szűri egy másik *ListBox*-ba.

18.18. ábra. Működés közben.



Magyarázat: A szűrőfeltételeket érdemes egy listában eltárolni (Egy int paramétert váró és logikai értéket visszaadó függvény *Predicate<int>* típusú.) Ezek után végig kell menni az összes elemen, és megnézni a szűrőfeltételeket (elég addig vizsgálni egy adott elemet, míg az egyik feltétel nem teljesül).

18.29. forráskód. Generálás és szűrés

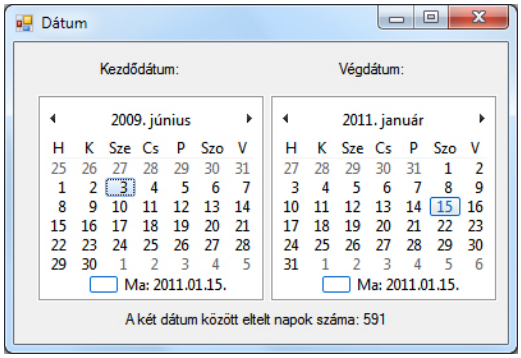
```
private void Btn_General_Click(object sender, EventArgs e)
{
  Btn_General.Enabled = false;
  Random rnd = new Random();
  HashSet<Int32> voltak = new HashSet<Int32>();
  LBx_Ossz.Items.Clear();
  LBx_Ossz.BeginUpdate();
  Int32 akt;
  for (Int32 i = 0; i < 1000; i++)
  {
    do
    {
```

```
        akt = rnd.Next(10000);
      } while (voltak.Contains(akt));
      voltak.Add(akt);
      LBx_Ossz.Items.Add(akt);
    }
    LBx_Ossz.EndUpdate();
    Btn_General.Enabled = true;
}

private void Btn_Mutat_Click(object sender, EventArgs e)
{ List<Predicate<Int32>> feltetelek = new List<Predicate<Int32>>();
  if (checkedListBox1.CheckedIndices.Contains(0))
     feltetelek.Add(Prim);
  if (checkedListBox1.CheckedIndices.Contains(1))
     feltetelek.Add(Paros);
  if (checkedListBox1.CheckedIndices.Contains(2))
     feltetelek.Add(Paratlan);
  if (checkedListBox1.CheckedIndices.Contains(3))
     feltetelek.Add(Oszt3);
  if (checkedListBox1.CheckedIndices.Contains(4))
     feltetelek.Add(Oszt6);
  if (checkedListBox1.CheckedIndices.Contains(5))
     feltetelek.Add(Oszt9);
  if (checkedListBox1.CheckedIndices.Contains(6))
     feltetelek.Add(Negyzet);
  LBx_Valogat.Items.Clear();
  for (Int32 i = 0; i < LBx_Ossz.Items.Count; i++)
  {
     Boolean kell = false;
     foreach (Predicate<Int32> feltetel in feltetelek)
     {
        kell = kell || feltetel((Int32)LBx_Ossz.Items[i]);
        if (kell)
           break;
     }
     if (kell)
        LBx_Valogat.Items.Add(LBx_Ossz.Items[i]);
  }
  LB_Eredm.Text = String.Format("{0}/{1}",
           LBx_Valogat.Items.Count, LBx_Ossz.Items.Count);
}
```

Egyéb eszközök, idő, dátum, érték beállítás

18.19. feladat (Napok száma – *szint: 1*). Írjon programot, melynek a segítségével bekér két dátumot, és meghatározza a két megadott dátum közötti napok számát!

18.19. ábra. Működés közben.

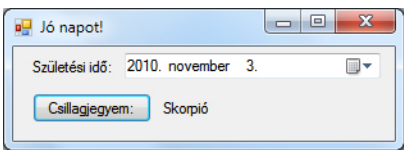


18.30. forráskód. Különbség kiszámítása

```
private void monthCalendar_DateChanged(object sender,
                              DateRangeEventArgs e)
{
   if (MC_kezd.SelectionStart.CompareTo(MC_veg.SelectionStart) > 0)
   {
      LB_kulonbseg.Text = "A keződátum nagyobb, mint a végdátum.";
   }
   else
   {
      LB_kulonbseg.Text =
         String.Format("A két dátum között eltelt napok száma: {0}",
         MC_veg.SelectionStart.Subtract(MC_kezd.SelectionStart).Days);
   }
}
```

18.20. feladat (Horoszkóp – *szint: 1*). Írjon programot, melynek a segítségével bekér egy születési dátumot, és meghatározza, hogy a felhasználó melyik csillagjegyben született. Az ablak fejlécében üdvözöljön minket napszaknak megfelelően.

18.20. ábra. Működés közben.



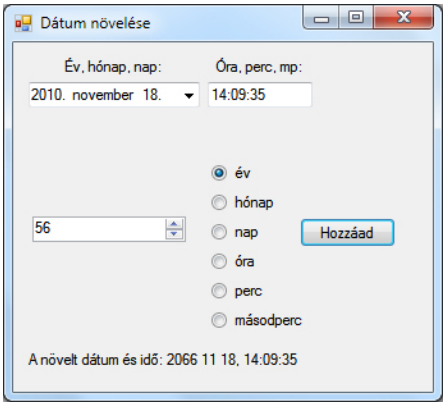
Magyarázat: Érdemes egy-egy tömbben a csillagjegyeket, és az azokat határoló dátumokat felsorolni. Ezek után a megadott dátum hónapjának és napjának figyelembevételével megkeresni melyik két határ közé esik.

18.31. forráskód. Csillagjegy megadása

```
private void Btn_Horoszkop_Click(object sender, EventArgs e)
{ label1.Text = Csillagjegy(dateTimePicker1.Value);}

private String Csillagjegy(DateTime dateTime)
{  dateTime = new DateTime(2011, dateTime.Month, dateTime.Day);
   DateTime[] dates = new DateTime[12] {
      new DateTime(2011, 01, 20),
      new DateTime(2011, 02, 19),
      new DateTime(2011, 03, 21),
      new DateTime(2011, 04, 20),
      new DateTime(2011, 05, 21),
      new DateTime(2011, 06, 22),
      new DateTime(2011, 07, 23),
      new DateTime(2011, 08, 23),
      new DateTime(2011, 09, 23),
      new DateTime(2011, 10, 23),
      new DateTime(2011, 11, 22),
      new DateTime(2011, 12, 22) };
   String[] jegyek = new String[12]{
      "Vízöntő","Halak","Kos",
      "Bika","Ikrek","Rák",
      "Oroszlán","Szűz","Mérleg",
      "Skorpió","Nyilas","Bak" };
   int i = 0;
   Boolean found = false;
   while (i < 11 && !found)
   {
      found = ((dates[i] < dateTime) && (dateTime < dates[i + 1]));
      i++;
   }
   return (found) ? jegyek[i - 1] : jegyek[11]; }
```

18.21. feladat (Időpont – *szint: 1*). Írjon programot, melynek a segítségével megnövelhető a dátum. Kiválasztható legyen melyik részét növelje a dátumnak vagy időnek (év, hónap, nap, óra, perc, másodperc).

18.21. ábra. Működés közben.



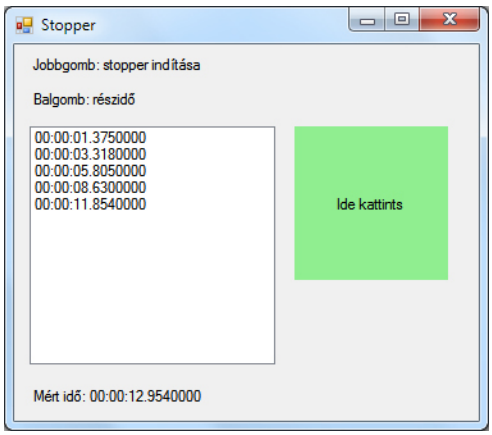
Magyarázat: Az óra, perc, másodperc megadását egy MaskedTextBox komponensben oldjuk meg "00:00:00" maszkkal. Figyeljen arra, hogy értelmezhető-e a megadott idő.

18.32. forráskód. Dátum növelése

```
private void BT_hozzaad_Click(object sender, EventArgs e)
{  DateTime ido;
   try
   { ido = DateTime.ParseExact(maskedTextBox1.Text, "HH:mm:ss",
                        CultureInfo.InvariantCulture);}
   catch (FormatException)
   { MessageBox.Show("Nincs megadva idő!"); return; }
   DateTime uj_datum_es_ido = new DateTime(
   datum.Value.Year, datum.Value.Month, datum.Value.Day,
   ido.Hour, ido.Minute, ido.Second );
   Int32 noveles = (int)numericUpDown1.Value;
   if (RB_ev.Checked)
      uj_datum_es_ido = uj_datum_es_ido.AddYears(noveles);
   else
      if (RB_honap.Checked)
         uj_datum_es_ido = uj_datum_es_ido.AddMonths(noveles);
      else
      if (RB_nap.Checked)
         uj_datum_es_ido = uj_datum_es_ido.AddDays(noveles);
      else
         if (RB_ora.Checked)
            uj_datum_es_ido = uj_datum_es_ido.AddHours(noveles);
         else
         if (RB_perc.Checked)
            uj_datum_es_ido = uj_datum_es_ido.AddMinutes(noveles);
         else if (RB_masodperc.Checked)
           uj_datum_es_ido = uj_datum_es_ido.AddSeconds(noveles);
   label3.Text = String.Format("A növelt dátum és idő:
            {0:yyyy MM dd, HH:mm:ss}", uj_datum_es_ido); }
```

18.22. feladat (Stopper – *szint: 1*). Írj stoppert, mely jobb egérgombbal indítja és állítja, bal egérgombbal pedig részeredményeket ad.

18.22. ábra. Működés közben.



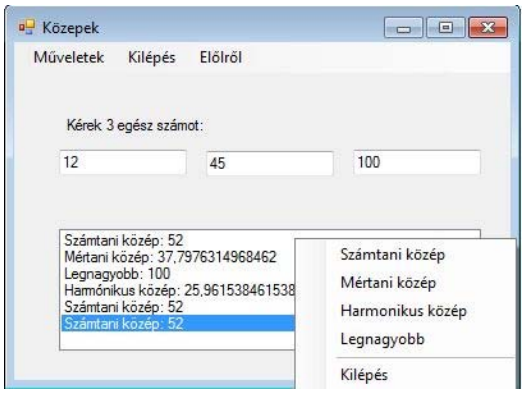
18.33. forráskód. Egérkattintás kezelése

```
private void Stopper_MouseClick(object sender, MouseEventArgs e)
{
   if (e.Button == System.Windows.Forms.MouseButtons.Right)
   {
      if (started)
      {  label1.Text = "Jobbgomb: stopper indítása";
         lb_mert.Text = String.Format("Mért idő: {0:t}",
                        (DateTime.Now - startTime)); }
      else
      {  label1.Text = "Jobbgomb: stopper leállítása";
         startTime = DateTime.Now;
         listBox1.Items.Clear();
         lb_mert.Text = String.Empty; }
      started = !started;
   }
   else if (e.Button == System.Windows.Forms.MouseButtons.Left)
   {
      if (started)
        listBox1.Items.Add((DateTime.Now - startTime).ToString("t"));
   }
}
```

Menük és eszköztárak

18.23. feladat (Három szám átlagai – *szint: 1*). Kérjünk be 3 egész számot. Különböző műveleteket lehessen végezni velük, melyek eredményét a művelet nevével egy ListBox-ba gyűjtse a program. A műveleteket lehessen a formon lévő menüsorból kiválasztani, illetve a ListBox területén megjelenő helyi menüből is. A műveletek: A 3 szám összege, a 3 szám számtani, mértani, harmonikus közepe. A legnagyobb szám a 3 közül.

18.23. ábra. Három szám átlagai.



Magyarázat: A menük és helyi menü használatánál hangoljuk össze a működést. Elég az eseménykezelőket egyszer megírni, pl. a menüpontoknál, és a helyimenü menüpontjainak eseménykezelőit irányítsuk ezekre a metódusokra. Használjuk a *MenuStrip* és *ContextMenuStrip* komponenseket. Ne felejtsük el a *ListBox*-nál a *ContextMenuStrip* tulajdonságot beállítani.

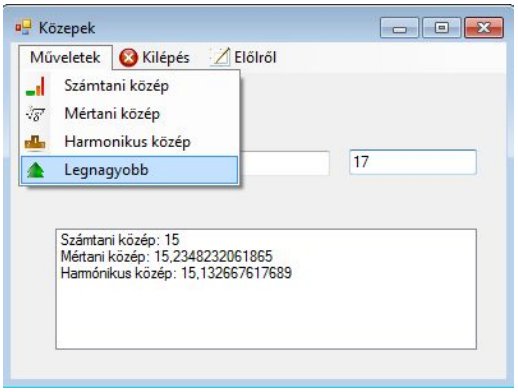
18.34. forráskód. Három szám mértani közepe

```
private bool JoSzamok()
{   return ((int.TryParse(textBoxSz1.Text, out sz1)) &&
         (int.TryParse(textBoxSz2.Text, out sz2)) &&
         (int.TryParse(textBoxSz3.Text, out sz3)));}

private void mértaniKözépToolStripMenuItem_Click(object sender,
                                    EventArgs e)
{   if (JoSzamok())
   { ListBoxEredmeny.Items.Add("Mértani közép: " +
               (Math.Pow(sz1 * sz2 * sz3, 1.0 / 3)).ToString()); }
   else
```

```
   { MessageBox.Show("3 egész számot kérek!!!", "Hiba...",
        MessageBoxButtons.OK, MessageBoxIcon.Error); } }
```

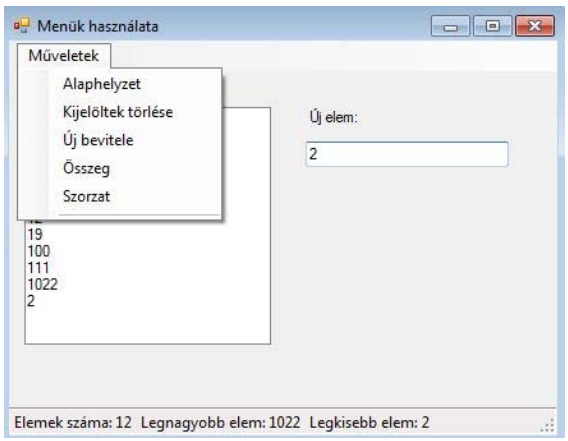
18.24. feladat (Menük ikonokkal – *szint: 2*). Egészítse ki az előző feladat megoldásául szolgáló programot úgy, hogy a menüpontok előtt kis ikonok is jelenjenek meg, melyek szimbolizálják a műveletet. Ugyanezek az ikonok jelenjenek meg egy eszköztáron is, ahonnan szintén elindítható legyen minden művelet.

18.24. ábra. Három szám átlagai ikonokkal.



18.25. feladat (Véletlen számok – *szint: 3*). Töltsön fel egy listát 10 és 20 közé eső véletlen számokkal. Legyen 10 elemű a lista. Lehessen a listából kijelölt elemeket törölni, új elemet bevinni, amit egy megjelenő TextBoxba tudunk bevinni, és lehessen a meglévő elemek összegét és szorzatát kiszámítani. Ezeket a műveleteket menüből kell indítani. A státuszsoron jelenítsük meg folyamatosan, hogy hány elemű a lista, mekkora a legkisebb és legnagyobb eleme.

18.25. ábra. Listaelemek kezekése menüből



Magyarázat: A feladat megoldásához alkalmazni kell a minimum-maximum kiválasztás tételét, az összegzés tételét alapesetben, és szorzatokra is. Ebben az esetben figyeljünk oda a gyűjtőváltozó kezdőérték adására. Az új érték bevitelénél csak egész számot fogadjunk el.

18.35. forráskód. Listaelemek kezelése

```
private void Szelsoertek(out int max, out int min)
{
   int i;
   min = max = Convert.ToInt32(listBox1.Items[0]);
   for (i = 1; i < listBox1.Items.Count; i++)
   {
      if (max < Convert.ToInt32(listBox1.Items[i]))
         max = Convert.ToInt32(listBox1.Items[i]);
      if (min > Convert.ToInt32(listBox1.Items[i]))
         min = Convert.ToInt32(listBox1.Items[i]);
   }
}

private void SetStatus()
{
   StatusLabel1.Text = "Elemek száma: " +
            listBox1.Items.Count.ToString();
   int max, min;
   Szelsoertek(out max, out min);
   StatusLabel2.Text = "Legnagyobb elem: " + max.ToString();
   StatusLabel3.Text = "Legkisebb elem: " + min.ToString();
}

private void SetDef()
{
   int i, szam;
   listBox1.Items.Clear();
   for (i = 0; i < 10; i++)
   {
      szam = rnd.Next(10) + 10;
      listBox1.Items.Add(szam.ToString());
   }
   SetStatus();
}
```

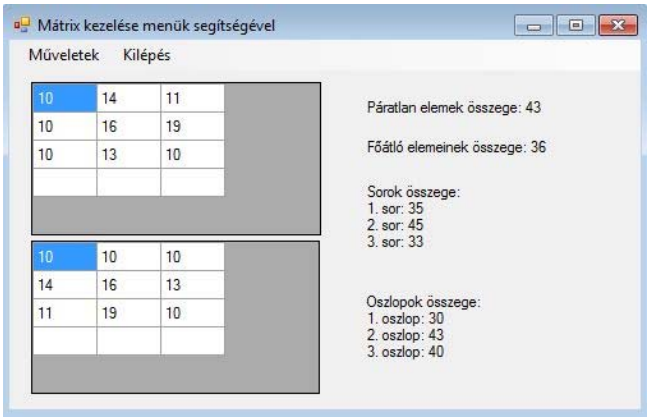
```
private void alaphelyzetToolStripMenuItem_Click(object sender,
                                    EventArgs e)
{
   SetDef();
}

private void összegToolStripMenuItem_Click(object sender,
                                    EventArgs e)
{
   int i, s = 0;
   for (i = 0; i < listBox1.Items.Count; i++)
   {
      s += Convert.ToInt32(listBox1.Items[i]);
   }
   labelOsszeg.Text = "Elemek összeg: " + s.ToString();
}
```

18.36. forráskód. Listaelemek kezelése

```
private void szorzatToolStripMenuItem_Click(object sender,
                                    EventArgs e)
{
   int i;
   double s = 1;
   for (i = 0; i < listBox1.Items.Count; i++)
   {
      s *= Convert.ToInt32(listBox1.Items[i]);
   }
   labelSzorzat.Text = "Elemek szorzata: " +
                  s.ToString();
}

private void kijelöltekTörléseToolStripMenuItem_Click(object sender,
                                        EventArgs e)
{
   while (listBox1.SelectedItems.Count > 0)
   {
      listBox1.Items.Remove(listBox1.SelectedItems[0]);
   }

   SetStatus();
}

private void újBeviteleToolStripMenuItem_Click(object sender,
                                    EventArgs e)
{
   int sz;
   if (int.TryParse(textBox1.Text, out sz))
            listBox1.Items.Add(textBox1.Text);

   SetStatus();
}
```

18.26. feladat (Véletlen számok mátrixban – *szint: 4*). Töltsön fel egy 3x3 mátrixot 10 és 20 közé eső véletlen számokkal. Végezze el menüből és helyi menüből választhatóan a következő műveleteket: transzponált, a páratlan számok összege, a főátló számainak összege, a sorokban lévő értékek és az oszlopokban lévő értékek összege. Ezeket az eredményeket jelenítse is meg a formon.

18.26. ábra. Mátrix elemek kezelése menüből



Magyarázat: A feladat megoldásánál használjuk a *DataGridView* komponens adta lehetőségeket. Figyeljünk arra, hogy a mátrix bejárásánál a sorokat és oszlopokat megfelelően azonosítsuk. Használjuk az összegzés tételét megfelelően. A transzponált előállítását lásd a 18.43-ben.

18.37. forráskód. Mátrix elemek generálása

```
private void General()
{
   Random rnd = new Random();
   int i, j, szam;
   dataGridAlap.Rows.Clear();
   for (i = 0; i < 3; i++)
   {
      DataGridViewRow r = new DataGridViewRow();
      for (j = 0; j < 3; j++)
      {
         szam = rnd.Next(10) + 10;
         DataGridViewCell dc = new DataGridViewTextBoxCell();
```

```
            dc.Value = szam;
            r.Cells.Add(dc);
        }
        dataGridAlap.Rows.Add(r);
    }
}
```

18.38. forráskód. Mátrix elemek kezelése

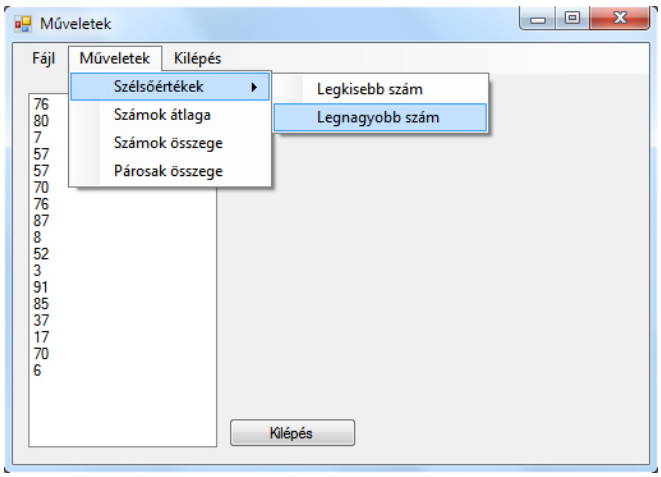
```
private void páratlanÖsszegToolStripMenuItem_Click(object sender,
                                        EventArgs e)
{
    int i, j, szam, ptl = 0;
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            szam = (int)dataGridAlap.Rows[j].Cells[i].Value;
            if (szam % 2 == 1) ptl += szam;
        }
    }
    labelPtlOsszeg.Text = "Páratlan elemek összege: " + ptl.ToString();
}

private void sorokÖsszegeToolStripMenuItem_Click(object sender,
                                        EventArgs e)
{
    int i, j, szam, s;
    labelSorok.Text = "Sorok összege: \n";
    for (i = 0; i < 3; i++)
    {
        s = 0;
        for (j = 0; j < 3; j++)
        {
            szam = (int)dataGridAlap.Rows[i].Cells[j].Value;
            s += szam;
        }
        labelSorok.Text = labelSorok.Text + (i+1).ToString() +
              ". sor: " + s.ToString() + "\n";
    }
}

private void főátlóÖsszegToolStripMenuItem_Click(object sender,
                                      EventArgs e)
{
    int i, szam, ossz = 0;
    for (i = 0; i < 3; i++)
    {
        szam = (int)dataGridAlap.Rows[i].Cells[i].Value;
        ossz += szam;
    }
    labelFoAtlOsszeg.Text = "Főátló elemeinek összege: " +
                                ossz.ToString();
}
```

18.27. feladat (Műveletek – *szint: 2*). Írj programot, amely egy számokkal töltött listában végez műveleteket. A műveleteket menüből érhessük el, és a következők legyenek:

- minimum/maximum keresés (ezt almenü segítségével old meg)
- átlagszámítás
- összeg
- páros számok összege

Ugyancsak menüből lehessen kiválasztani, hogy kézzel adjuk meg a számokat, vagy a program generálja őket. Menüből és gomb segítségével is lehessen kilépni a programból. A számok bekérésénél végezzünk ellenőrzést, hogy valóban egész számot adjon meg a felhasználó és a megadott határokon belül legyen: 1 és 20 között lehet a darabszám, és a számok 1 és 99 között legyenek a listában.

18.27. ábra. Működés közben



Magyarázat: Mivel gyakran kell egy bekért szám ellenőrzését elvégezni, írjunk rá külön metódust, mely egy *textbox* szövegét próbálja meg átalakítani és megadható

neki két határ, amik közé kell essen a szám.

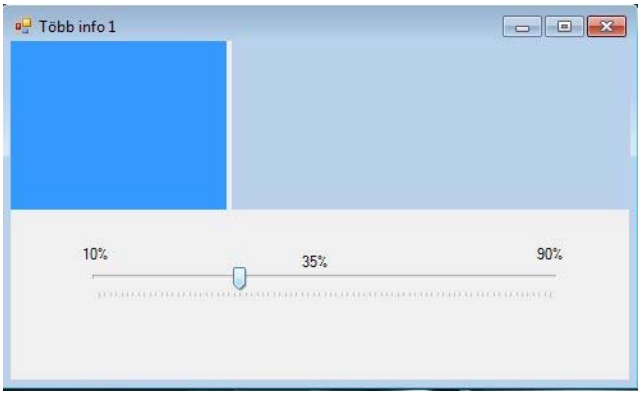
18.39. forráskód. Ellenőrzés és összegzés

```
private Boolean Ellenoriz(TextBox tb, Int32 ah, Int32 fh, out Int32 szam)
{
   if (Int32.TryParse(tb.Text, out szam) &&
      (szam >= ah) && (szam <= fh))
   {
      return true;
   }
   else
   {
      MessageBox.Show(String.Format("Számot kell megadni és {0},"
                        +"{1} között kell lennie!", ah, fh));
      tb.Text = String.Empty;
      tb.Focus();
      return false;
   }
}

private void párosakÖsszegeToolStripMenuItem_Click(object sender,
                                    EventArgs e)
{
   Int32 sum = 0;
   for (Int32 i = 0; i < listBox.Items.Count; i++)
   {
      if ((Int32)listBox.Items[i] % 2 == 0)
      {
         sum += (Int32)listBox.Items[i];
      }
   }
   if (sum > 0)
     MessageBox.Show(String.Format("A páros számok összege: {0}"
                           , sum));
   else
     MessageBox.Show("Nincs páros szám!");
}

private void számokátlagaToolStripMenuItem_Click(object sender,
                        EventArgs e)
{
   Int32 sum = 0;
   for (Int32 i = 0; i < listBox.Items.Count; i++)
   {
     sum += (Int32)listBox.Items[i];
   }
   MessageBox.Show(String.Format("A számok átlaga: {0}", sum /
            ((double)listBox.Items.Count)));
}
```

Több info egy formon

18.28. feladat (SplitContainer használata – *szint: 1*). Írjon programot, mely tartalmaz 2 színes panelt egymás mellett, és lehetőség legyen a területük arányának változtatására. Adjuk meg a jobboldali panel szélességének arányát az egészhez képest.

18.28. ábra. Működés közben a SplitContainer.



Magyarázat: A panelek eltolására használja a SplitContainer komponenst, míg látványosan az arány megaddható a Trackbar segítségével.

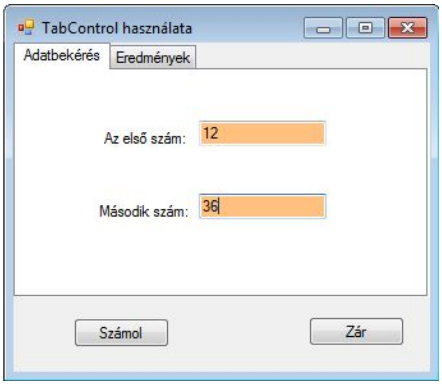
18.40. forráskód. SplitContainer használata

```
namespace Tobbinfo_1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void trackBar1_Scroll(object sender, EventArgs e)
      {
         splitContainer2.SplitterDistance =
            splitContainer2.Width * trackBar1.Value / 100;
         labelPos.Text = trackBar1.Value.ToString() + "%";
      }
```

```
      private void Form1_Shown(object sender, EventArgs e)
      {
         splitContainer2.SplitterDistance =
            splitContainer2.Width * 10 / 100;
         labelPos.Text = trackBar1.Value.ToString() + "%";
      }
   }
}
```

18.29. feladat (TabControl használata – *szint: 1*). Írjon programot, mely bekér két egész számot külön külön egy TabControl egyik lapján. Majd kiszámolja a két szám számtani közepét, amit kiír egy másik lapra, aminek a címkéje legyen: *eredmények*. A bekéréskor ügyeljen a kivételkezelésre! A programból való kilépéskor kérdezzen rá, hogy biztos ki akar-e lépni a felhasználó, és a válasznak megfelelően járjon el.

18.29. ábra. Adatbekérés TabControl használatával.



Magyarázat: Figyeljünk oda a hibakezelésre, és a váltásra a *TabPage*-ek között.

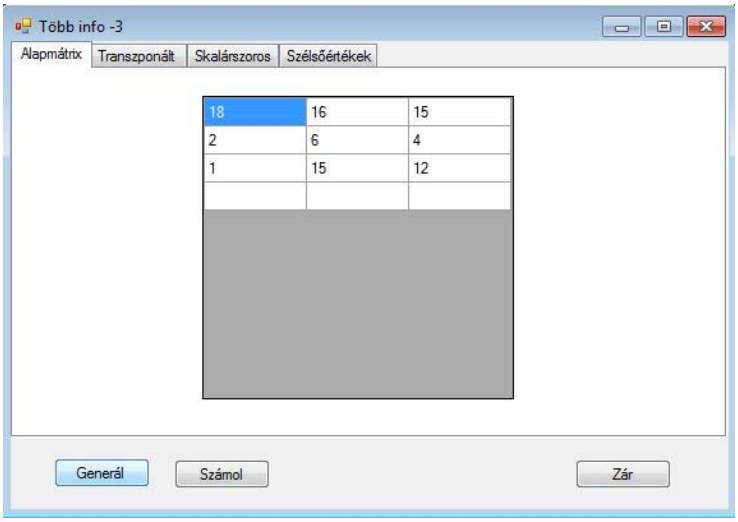
18.41. forráskód. TabControl használata

```
private void buttonSzamol_Click(object sender, EventArgs e)
{
  int szam1, szam2;
  if (textBox1.Text != String.Empty &&
     textBox2.Text != String.Empty)
  {
  try
  {
     szam1 = Convert.ToInt32(textBox1.Text);
     szam2 = Convert.ToInt32(textBox2.Text);
     labelEredmeny.Text = ((szam1 + szam2) / 2).ToString();
     tabControl1.SelectTab(1);
  }
  catch
  {
     MessageBox.Show("Adathiba!", "Hiba",
         MessageBoxButtons.OK, MessageBoxIcon.Error);
  }
  }
  else
  {
    MessageBox.Show("Mindkét számot írd be!", " Hiba",
         MessageBoxButtons.OK, MessageBoxIcon.Error);
  }
}
```

18.30. feladat (Mátrixok adatai egy formon – *szint: 3*). Írjon programot, mely generál egy 3x3 mátrixot olyan véletlen számokból, melyek 1 és 20 közé esnek. Határozza meg a mátrix transzponáltját, a skalárszorosát és a mátrix legkisebb és legnagyobb értékét is válassza ki. Az egyes mátrixokat és a szélsőértékeket külön lapokon jelenítse meg, de csak egy formot használjon.

Magyarázat: A kezdeti állapotban a program generáljon egy kiinduló mátrixot. Figyeljen oda, hogy a véletlen számok a megadott intervallumba essenek. Megjelenítéshez lehet használni a *DataGridView* egy megfelelően beállított példányát.

18.30. ábra. Kiinduló mátrix.

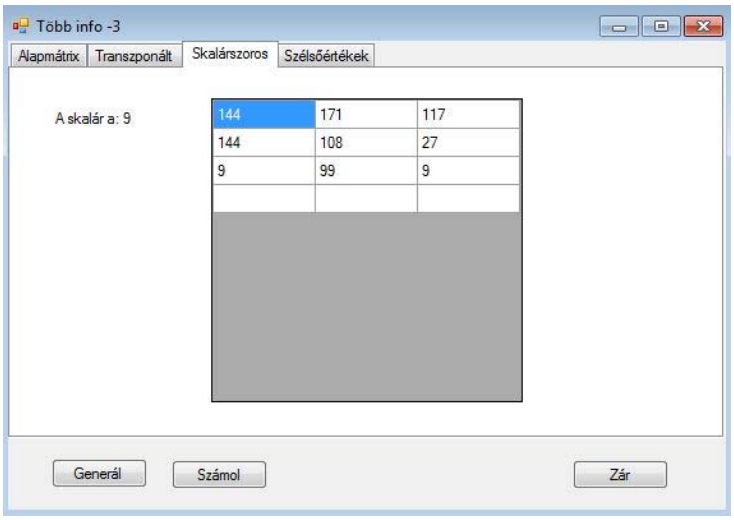


18.42. forráskód. Kiinduló mátrix generálása

```
private void buttonAlap_Click(object sender, EventArgs e)
{
  Random rnd = new Random();
  int i, j, szam;
  dataGridAlap.Rows.Clear();
  for (i = 0; i < 3; i++)
  {
     DataGridViewRow r = new DataGridViewRow();
     for (j = 0; j < 3; j++)
     {
       szam = (rnd.Next() % 20) + 1;
       DataGridViewCell dc = new DataGridViewTextBoxCell();
       dc.Value = szam;
       r.Cells.Add(dc);
     }
     dataGridAlap.Rows.Add(r);
  }
}
```

Magyarázat: A számolás indításakor határozza meg a feladatban előírt mátrixokat és számokat.

18.31. ábra. Kiinduló mátrix.



18.43. forráskód. Transzponált számítása

```
private void Transzponalt()
{
 int i, j, szam;
 dataGridTr.Rows.Clear();
 for (i = 0; i < 3; i++)
 {
   DataGridViewRow r = new DataGridViewRow();
   for (j = 0; j < 3; j++)
   {
      szam = (int)dataGridAlap.Rows[j].Cells[i].Value;
      DataGridViewCell dc = new DataGridViewTextBoxCell();
      dc.Value = szam;
      r.Cells.Add(dc);
   }
   dataGridTr.Rows.Add(r);
        }
     }
```

18.44. forráskód. Skalárral szorzás

```
private void SkalarSzorzas()
{
 Random rnd = new Random();
 int i, j, szam, szorzo;
 szorzo = (rnd.Next() % 7) + 3;
 labelSkalar.Text = " A skalár a: "+szorzo.ToString();
 dataGridSzor.Rows.Clear();
 for (i = 0; i < 3; i++)
 {
   DataGridViewRow r = new DataGridViewRow();
   for (j = 0; j < 3; j++)
   {
     szam = (int)dataGridAlap.Rows[j].Cells[i].Value * szorzo;
     DataGridViewCell dc = new DataGridViewTextBoxCell();
     dc.Value = szam;
     r.Cells.Add(dc);
   }
   dataGridSzor.Rows.Add(r);
  }
}
```

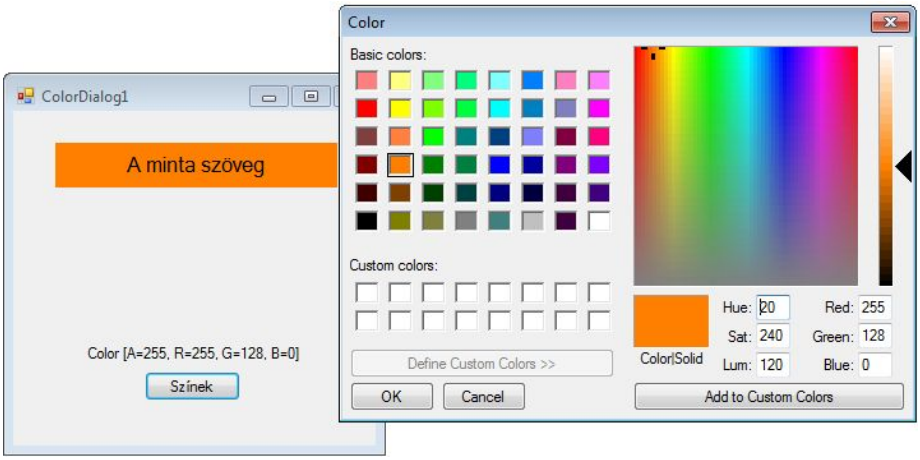
18.45. forráskód. Szélsőértékek

```
private void SzelsoErtekek()
     {
         int szam, i,j, min = 21, max = -1;
         for (i = 0; i < 3; i++)
         {
            for (j = 0; j < 3; j++)
            {
               szam = (int)dataGridAlap.Rows[i].Cells[j].Value;
               if (min > szam) min = szam;
               if (max < szam) max = szam;
            }
         }
         labelMin.Text = min.ToString();
         labelMax.Text = max.ToString();
      }
```

Dialógusok

18.31. feladat (Színek állítása dialógusablak segítségével. – *szint: 1*). Készítsen egy olyan programot, mely egy formon lévő *label*-nek a szín tulajdonságait állítja. A színbeállítást egy nyomógomb segítségével kezdeményezze. A színeket a szokásos windows beállító ablak segítségével lehessen kiválasztani. Egy további *label*-be jelenítsük meg a kiválaszott szín adatait is.

18.32. ábra. Működés közben.



Magyarázat: A *ColorDialog* példányosítása történhet a komponens használatával, vagy a kódból közvetlenül is! Figyeljen arra, hogy a *ColorDialog* is, mint minden dialógus ablak, a *ShowDialog()* metódussal hívható, melynek visszatérési értéke a *DialogResult* osztály egy példányában fogadható, és utána értékelhető ki.

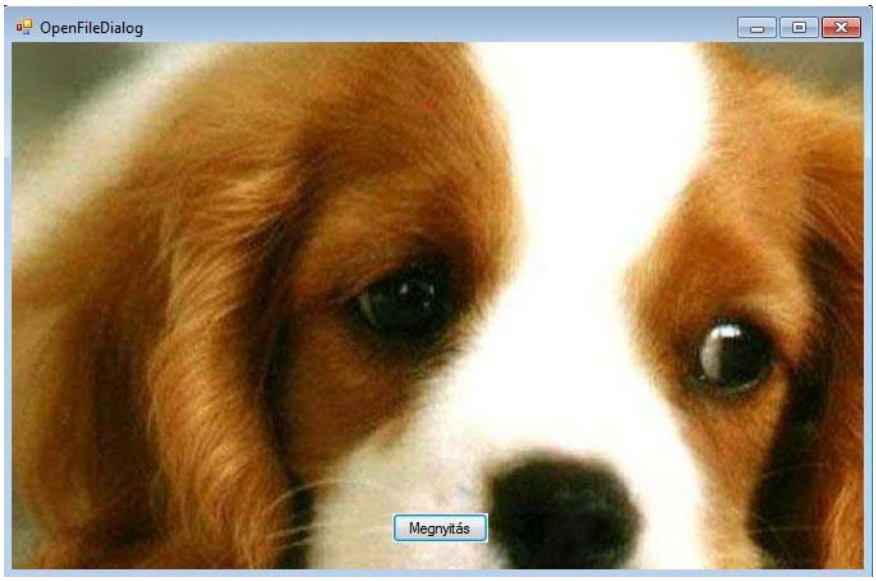
18.46. forráskód. ColorDialog használata

```
private void buttonColors_Click(object sender, EventArgs e)
{
   ColorDialog cd = new ColorDialog();
   DialogResult dr;
   dr = cd.ShowDialog();
   if (dr == DialogResult.OK)
   {
      labelMinta.BackColor = cd.Color;
      labelVal.Text = cd.Color.ToString();
   }
}
```

18.32. feladat (Kép megnyitása dialógus ablak segítségével. – *szint: 1*). Készítsen egy olyan programot, melyben a szokásos windows Megnyitás ablak

segítségével kiválaszt egy képet, és azt megnyitja egy *PictureBox*-ban. A kép töltse ki a *form* felületét, méretezésre kövesse annak mozgását. A megnyitást egy gomb segítségével kezdeményezze.

18.33. ábra. Kép megnyitása.



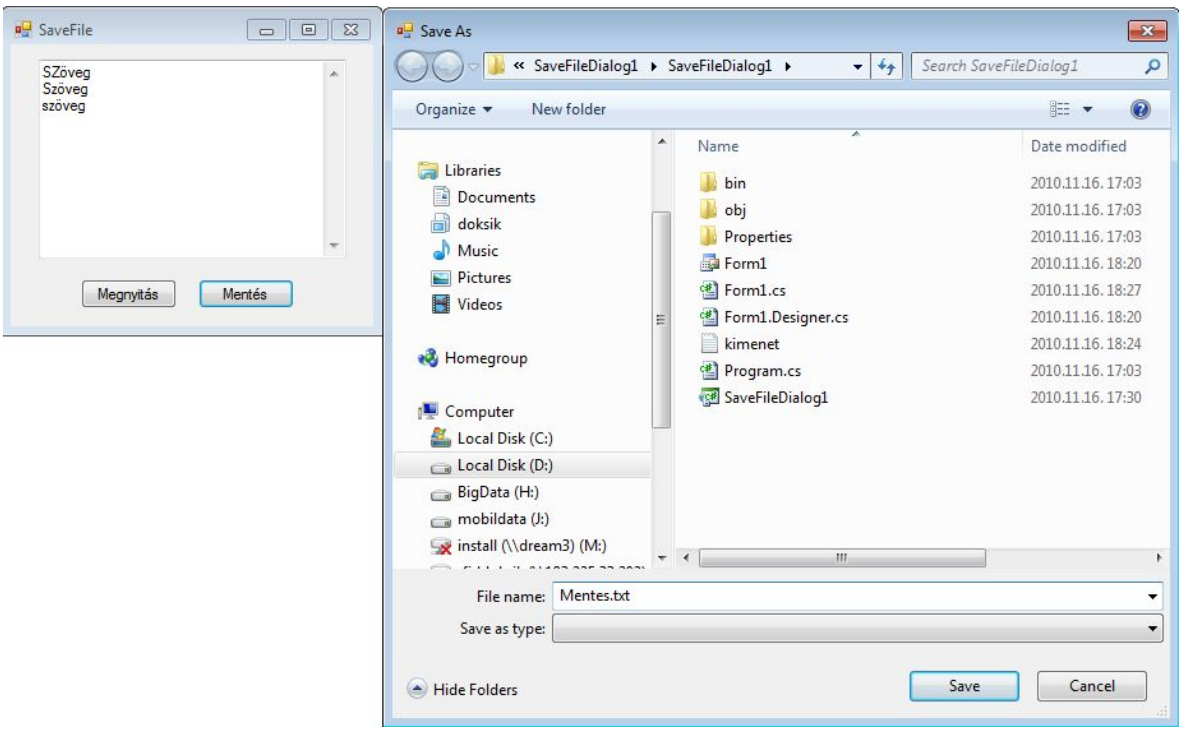
Magyarázat: Figyeljen rá, hogy a *PictureBox Dock* tulajdonságát hogy állítja be!

18.47. forráskód. OpenFileDialog használata

```
private void buttonOpen_Click(object sender, EventArgs e)
      {
         OpenFileDialog of = new OpenFileDialog();
         DialogResult dr = of.ShowDialog();
         if (dr == DialogResult.OK)
         {
            pictureBoxDest.SizeMode = PictureBoxSizeMode.CenterImage;
            pictureBoxDest.Image = new Bitmap(of.FileName);
         }
      }
```

18.33. feladat (Szöveges file megnyitása és mentése – *szint: 2*). Készítsen egy programot, mely a szokásos windows Megnyitás ablak segítségével meg tud nyitni egy szöveges file-t egy *TextBox*-ba. Illetve tudjuk elmenteni a tartalmát a szokásos Mentés ablak használatával.

18.34. ábra. Szöveges file mentése dialógus ablakkal.



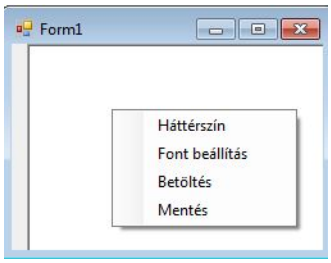
Magyarázat: Figyeljen oda, hogy a *TextBox Multiline* tulajdonságát hogy állítja be. Ne feledjük, a file-ok használathához a *System.IO* névtérre szükség van.

18.48. forráskód. A SaveFileDialog használata

```
private void buttonOpen_Click(object sender, EventArgs e)
{ OpenFileDialog of = new OpenFileDialog();
 if (of.ShowDialog() == DialogResult.OK)
 { FileStream fs = new FileStream(of.FileName,
                 FileMode.Open);
   StreamReader rs = new StreamReader(fs);
   string s = rs.ReadLine();
   while (s != null)
   { textBoxDest.Text += s;
    s = rs.ReadLine(); }
   rs.Close(); fs.Close(); } }

private void buttonSave_Click(object sender, EventArgs e)
{ SaveFileDialog sf = new SaveFileDialog();
 if (sf.ShowDialog() == DialogResult.OK)
 { FileStream fs = new FileStream(sf.FileName,
           FileMode.Create);
   StreamWriter wr = new StreamWriter(fs);
   wr.Write(textBoxDest.Text);
   wr.Close(); fs.Close(); } }
```

18.34. feladat (Dialógusok használata – *szint: 3*). Készítsen programot mely egy *RichText* komponensbe megnyit egy kiválasztott rtf file-t. Tudjuk módosítani a háttér színét, a karakterek színét, a karakterek jellemzőit. Tudjunk menteni. Az egyes lehetőségeket helyi menüből érjük el.

18.35. ábra. Helyi menü használata a feladatban.



Magyarázat: A háttérszín beállításánál használjuk a *ColorDialog* komponenst. A karakterek jellemzőit a FontDialog komponens segítségével állíthatjuk be. Ennek a Font tulajdonsága tartalmazza az összes beállítást.

18.49. forráskód. Háttérszín és Font beállítása

```
private void háttérszínToolStripMenuItem_Click(object sender,
                                   EventArgs e)
{ ColorDialog cd = new ColorDialog();
  if (cd.ShowDialog() == DialogResult.OK)
    richTextBox1.BackColor = cd.Color; }

private void fontBeállításToolStripMenuItem_Click(object sender,
                                   EventArgs e)
{ FontDialog fd = new FontDialog();
   if (fd.ShowDialog() == DialogResult.OK)
       richTextBox1.Font = fd.Font; }
```

Magyarázat: Mentésnél figyeljünk arra, hogy file művelet végzésekor erőforrást használunk. Szükség lehet a kivételkezelésre.

18.50. forráskód. Mentés és visszatöltés

```
private void betöltésToolStripMenuItem_Click(object sender, EventArgs e)
{ OpenFileDialog of = new OpenFileDialog();
 if (of.ShowDialog() == DialogResult.OK)
   richTextBox1.LoadFile(of.FileName); }

private void mentésToolStripMenuItem_Click(object sender, EventArgs e)
{ SaveFileDialog sf = new SaveFileDialog();
 if (sf.ShowDialog() == DialogResult.OK)
    richTextBox1.SaveFile(sf.FileName); }
```

Modális és nem modális formok

18.35. feladat (Üzenetablak használata – *szint: 1*). Írjon programot, mely véletlenszerűen kitalál egy egész számot 1 és 20 között. A felhasználó tippelje meg, hogy páros vagy páratlan lesz. A kiértékelést egy dialógus üzenettel végezze a program.

18.36. ábra. Kiértékelés üzenetablak segítségével.

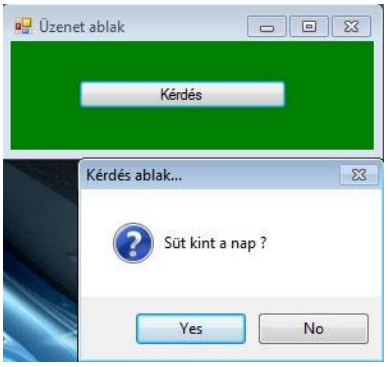
Magyarázat: Ügyeljen a véletlenszámok használatára. Az üzenetablak a *MessageBox Show* metódusával hívható, melynek 4 paramétere van. A megjelenítendő szöveg, a form fejlécszövege, a megjelenő gombok, és az ikon. A gombok és az ikon a rendszer felsorolt típusa *MessageBoxButtons* és *MessageBoxIcon*. Ezek közül kell választani. A Páratlan tipp kiértékelése teljesen hasonlóan mehet.

18.51. forráskód. Deklarálás

```
private int aSzam = 0;
private bool ParosE;
private Random rnd = new Random();

private void buttonGen_Click(object sender, EventArgs e)
{ aSzam = (rnd.Next() % 20) + 1;
  ParosE = ((aSzam % 2) == 0);
  labelSZAM.Text = "Generálás kész"; }

private void buttonParos_Click(object sender, EventArgs e)
{  if (ParosE && aSzam != 0)
        MessageBox.Show("Jól tippeltél nagyon. ", "Eredmény",
          MessageBoxButtons.OK, MessageBoxIcon.Information);
   else
      MessageBox.Show("Rossz tipp. ", "Eredmény",
           MessageBoxButtons.OK, MessageBoxIcon.Warning);
   labelSZAM.Text = aSzam.ToString(); }
```

18.36. feladat (Üzenetablak kiértékelése – *szint: 1*). Írjon programot, mely feltesz egy eldöntendő kérdést egy üzenet ablakban, és a válasznak megfelelően, ha IGEN, akkor zöldre, ha NEM akkor pirosra színezi a form hátterét.

18.37. ábra. Üzenetablak gombjai.

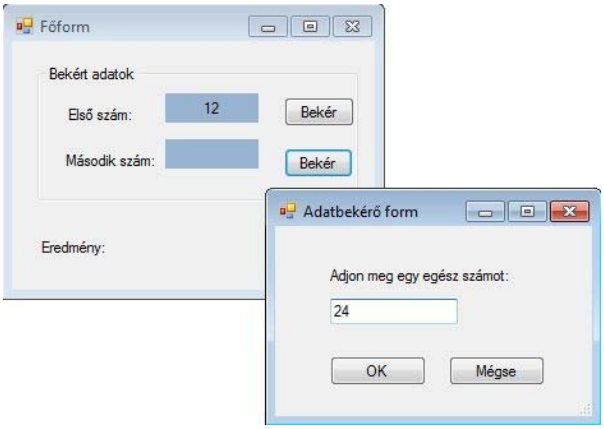


18.52. forráskód. Egy lehetséges megoldás.

```
private void buttonKerdes_Click(object sender, EventArgs e)
{
  if (MessageBox.Show("Süt kint a nap ?", "Kérdés ablak...",
     MessageBoxButtons.YesNo,
      MessageBoxIcon.Question) == DialogResult.Yes)
  {
     this.BackColor = Color.Green;
  }
  else
  {
     this.BackColor = Color.Red;
  }
}
```

18.37. feladat (Adatbekérés modális form segítségével. – *szint: 3*). Írjon programot, mely bekér két egész számot külön külön egy adatbekérő modális form segítségével. Majd kiszámolja a két szám számtani átlagát, amit kiír a főformra. A bekéréskor ügyeljen a kivételkezelésre! A programból való kilépéskor kérdezzen rá, hogy biztos ki akar-e lépni a felhasználó, és a válasznak megfelelően járjon el.

18.38. ábra. Modális form.



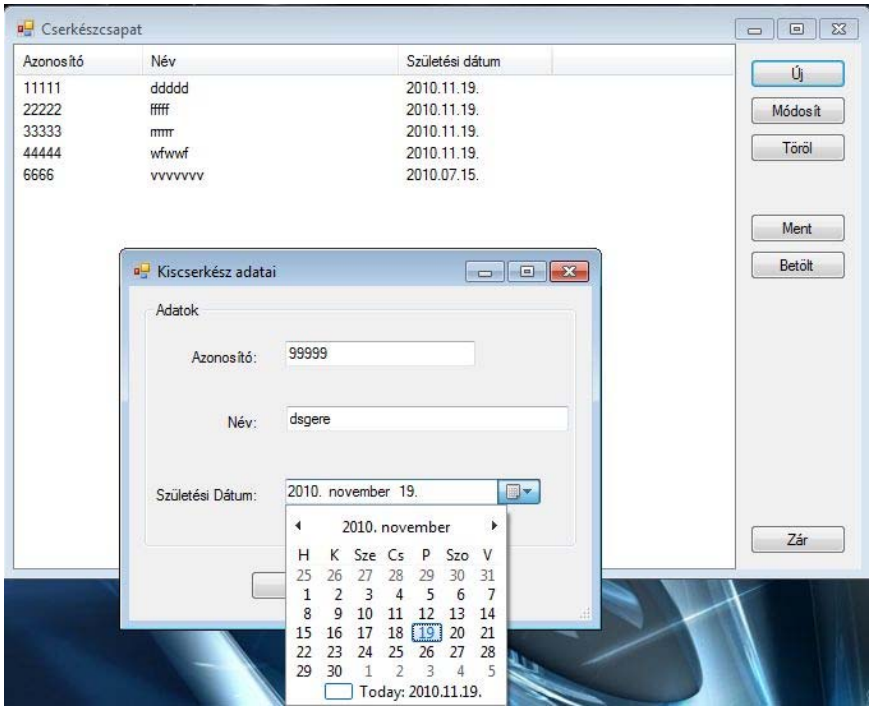
Magyarázat: Figyeljen oda, hogy az adatbekérés után publikus adatmezőbe kerüljön az adat, hogy át lehessen venni a másik formból. Ne felejtse el az adatbekérő formon a felrakott gombok *DialogResult* értékét beállítani. Alapértelmezése *None*. Az adatbekéréshez példányosítsuk a bekérő formot. A kilépésnél kezeljük a MessageBox Show metódusának a visszatérési értékét.

18.53. forráskód. Egy lehetséges megoldás.

```
private void buttonBe1_Click(object sender, EventArgs e)
{
   FormBeker frm = new FormBeker();
   if (frm.ShowDialog() == DialogResult.OK)
   {  SzamEgy = frm.BekertSzam;
      labelElso.Text = frm.BekertSzam.ToString(); }
}

private void FormMain_FormClosing(object sender, FormClosingEventArgs e)
{
   if (MessageBox.Show("Biztos ki akar lépni?", "Kérdés",
      MessageBoxButtons.OKCancel, MessageBoxIcon.Question) ==
      DialogResult.Cancel)
   {  e.Cancel = true;  } }
```

18.38. feladat (Összetartozó adatok kezelése – *szint: 5*). Készítsen egy programot, melyben egy kiscserkész csapat tagjainak adatait tudjuk nyilvántartani. A tagokról az azonosítójukat, a nevüket, és a születési dátumukat kell letárolni. Az adatbevitel egy modális form segítségével történjen. Az adatokat táblázatos formában jelenítsük meg. Tudjunk adatot menteni és visszatölteni fileból, a szokásos windows dialógus ablakok használatával.

18.39. ábra. Kiscserkészek adatai



Magyarázat: A feladat során több mindenre kell figyeljünk. Amit át kell tekinteni, a ListView kezelése, bináris file írása és olvasása, modális form használata, és adatforgalom a formok között.

Tekintsük először az adatbeviteli formot. Itt csak arra ügyeljünk, hogy az adatmezőket publikus láthatóságra állítsuk, hogy a *FormMain* osztályból is el lehessen érni.

18.54. forráskód. Adatbeviteli form

```
// Az adatbeviteli form

public partial class FormAdat : Form
{
```

```
  public string Beazon = "";
  public string Benev = "";
  public DateTime Beszul;

  public FormAdat()
  {
    InitializeComponent();
  }

  private void buttonOK_Click(object sender, EventArgs e)
  {
    Beazon = textBoxAzon.Text;
    Benev = textBoxNev.Text;
    Beszul = dateTimePickerSzDatum.Value;
  }
}
```

Magyarázat: A FormMain osztály eseménykezelői kissé összetettebbek, mert itt végezzük el a feladatokat.

18.55. forráskód. A főform eseménykezelői, metódusai

```
// A főform

public partial class FormMain : Form
{
   private string azon = "";
   private string nev = "";
   private DateTime szul;

   public FormMain()
   {
      InitializeComponent();
   }

   private void buttonZar_Click(object sender, EventArgs e)
   {
      Close();
   }

   private void FormMain_FormClosing(object sender,
                 FormClosingEventArgs e)
   {
      if (MessageBox.Show(" Biztos kilép?", "Kérdés",
         MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==
                           DialogResult.No)
      {
         e.Cancel = true;
      }
   }
```

18.56. forráskód. A főform eseménykezelői, metódusai

```
   // Kiválasztott adat módosítása
private void buttonMod_Click(object sender, EventArgs e)
{
   FormAdat frm = new FormAdat();
   frm.textBoxAzon.Text =
  listViewData.Items[listViewData.SelectedIndices[0]].Text;
   frm.textBoxNev.Text =
  listViewData.Items[listViewData.SelectedIndices[0]].SubItems[1].Text;
   DateTime d = new DateTime();
   d = Convert.ToDateTime(
  listViewData.Items[listViewData.SelectedIndices[0]].SubItems[2].Text);
   frm.dateTimePickerSzDatum.Value = d;
   if (frm.ShowDialog() == DialogResult.OK)
   {
  listViewData.Items[listViewData.SelectedIndices[0]].Text =
        frm.Beazon;
  listViewData.Items[listViewData.SelectedIndices[0]].SubItems[1].Text =
        frm.Benev;
  listViewData.Items[listViewData.SelectedIndices[0]].SubItems[2].Text =
        frm.Beszul.ToShortDateString();
   }
}

   // Bináris file-ba mentés
   private void buttonSave_Click(object sender, EventArgs e)
   {
      SaveFileDialog sf = new SaveFileDialog();
      if (sf.ShowDialog() == DialogResult.OK)
      {
         BinaryWriter br = new BinaryWriter(File.Open(sf.FileName,
                                FileMode.Create));
         try
         {
            int i;
            for (i = 0; i < listViewData.Items.Count; i++)
            {
               br.Write(listViewData.Items[i].Text);
               br.Write(listViewData.Items[i].SubItems[1].Text);
               br.Write(listViewData.Items[i].SubItems[2].Text);
            }
            br.Flush();
         }
         catch
         {
```

```
            MessageBox.Show("Hiba a mentésben.");
        }
        finally
        {
            br.Close();
        }
    }
}
```

18.57. forráskód. A főform eseménykezelői, metódusai

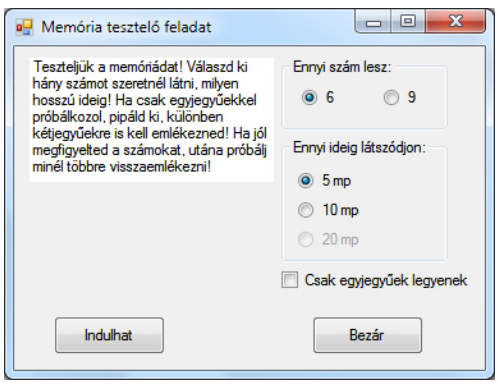
```
// Visszatöltés bináris file-ból
private void buttonOpen_Click(object sender, EventArgs e)
{
   OpenFileDialog of = new OpenFileDialog();
   if (of.ShowDialog() == DialogResult.OK)
   {
      string sa;
      listViewData.Items.Clear();
      BinaryReader br = new BinaryReader(File.Open(of.FileName,
                                FileMode.Open));
      try
      {
         if (File.Exists(of.FileName))
         {
            long len1 = br.BaseStream.Length;
            while (br.BaseStream.Position < len1)
            {
               azon = br.ReadString();
               nev = br.ReadString();
               sa = br.ReadString();
               ListViewItem li = new ListViewItem(azon, 0);
               li.SubItems.Add(nev);
               li.SubItems.Add(sa);
               listViewData.Items.Add(li);
            }
         }
      }
      catch { }
      finally
      {
         br.Close();
      }
   }
}

// Sor törlése
private void buttonDel_Click(object sender, EventArgs e)
{
   listViewData.Items[listViewData.SelectedIndices[0]].Remove();
}
```

18.39. feladat (Memória – *szint: 4*). Írj memória játék programot. A következő beállításokat támogassa a program:

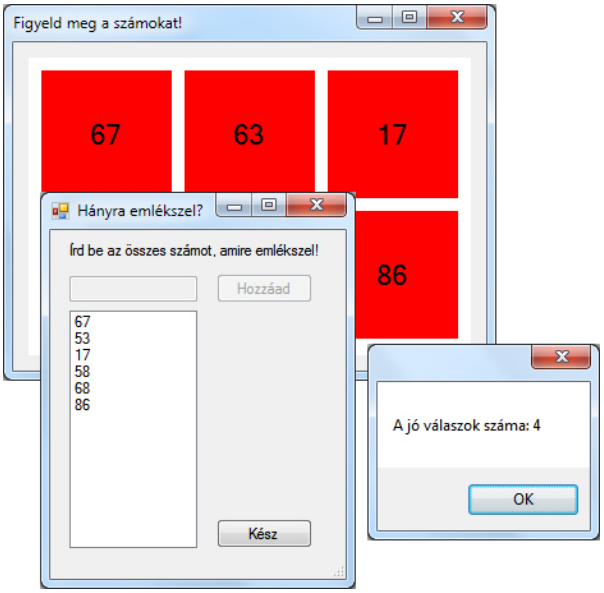
- hány szám legyen (6 vagy 9)
- mennyi ideig látszódjanak (5, 10, vagy 20 mp)
- hány jegyűek legyenek (1 vagy 2)

Ha 6 db számot kell kitalálni, akkor 5/10 mp, ha 9 számot, akkor pedig 10/20 mp legyen a választható. A program a beállított paramétereknek megfelelően generáljon számokat és jelenítse meg azokat egy ideig, majd kérdezze vissza őket. A számok megjelenítéséhez és visszakérdezéséhez is modális ablakokat használjunk. A visszakérdezéskor csak annyi tippje lehessen a játékosnak, mint amennyi a kitalálandó számok darabszáma.

18.40. ábra. Kezdő ablak



18.41. ábra. További megjelenő formok



18.58. forráskód. Játék indítása és kiértékelése

```
private void BT_Indulhat_Click(object sender, EventArgs e)
{
   int db = (RB_szam6.Checked) ? 6 : 9;
   Frm_Szamok frm_Szamok = new Frm_Szamok(
      db,
      (CB_Egyjegyu.Checked) ? 1 : 2,
      RB_mp5.Checked ? 5 : RB_mp10.Checked ? 10 : 20);
   frm_Szamok.ShowDialog();
   Frm_Memoria frm_Memoria = new Frm_Memoria(db);
   frm_Memoria.ShowDialog();
   int joValasz = 0;
   foreach (int i in frm_Memoria.tippek)
   {
      if (frm_Szamok.szamok.Contains(i))
      {
         joValasz++;
      }
   }
   MessageBox.Show(String.Format("A jó válaszok száma: {0}", joValasz));
}
```

18.59. forráskód. Számok megjelenítése

```
private void LabelekKirak(int db, int szamjegy)
{  Random veletlen = new Random();
   int meretSzelesseg = (panel1.Width - 40) / 3;
   int meretMagassag = 200 / (db / 3);
   szamok = new List<int>(db);
   int min, max;
   if (szamjegy == 1)
   {  min = 1; max = 9; }
   else
   {  min = 10; max = 99; }
   for (int i = 0; i < db; i++)
   {
      Label ujLabel = new Label();
      int x = i % 3, y = i / 3;
      ujLabel.Location = new Point(x * (meretSzelesseg + 10) + 10,
                                   y * (meretMagassag + 10) + 10);
      ujLabel.Size = new Size(meretSzelesseg, meretMagassag);
      ujLabel.Font = new Font(Font.FontFamily, 16);
      ujLabel.TextAlign = ContentAlignment.MiddleCenter;
      ujLabel.BackColor = Color.Red;
      int ujSzam;
      do
      {  ujSzam = veletlen.Next(max - min + 1) + min;
      } while (szamok.Contains(ujSzam));
      szamok.Add(ujSzam);
      ujLabel.Text = ujSzam.ToString();
      panel1.Controls.Add(ujLabel);
   }
}
```

Magyarázat: A labeleket dinamikusan rakja fel a formra, mivel nem tudni előre, hogy hány számot kell megjeleníteni.

18.60. forráskód. Tipp hozzáadása

```
private void BT_Hozzaad_Click(object sender, EventArgs e)
{
   if (!listBox1.Items.Contains(TB_Szam.Text))
   {
      int temp;
      if (int.TryParse(TB_Szam.Text, out temp))
      {
         listBox1.Items.Add(TB_Szam.Text);
         tippek.Add(temp);
```

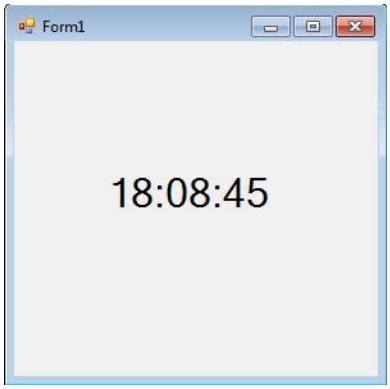
```
            }
        }
        TB_Szam.Text = String.Empty;
        TB_Szam.Focus();
        if (maxDb == listBox1.Items.Count)
        {
            TB_Szam.Enabled = BT_Hozzaad.Enabled = false;
        }
    }
```

Magyarázat: Maximálisan annyi számot lehessen beírni, amennyit eredetileg kiválasztottunk megtekintésre.

Időzítés és üzenetek

18.40. feladat (Időzítés használata – *szint: 2*). Írj programot, mely a form közepén mutatja a futó időt másodperc pontosan.

18.42. ábra. Időzítés használata



Magyarázat: Az időzítés legegyszerűbben a Timer komponens segítségével oldhatjuk meg. Használjuk a Tick eseményt. Figyeljünk rá, hogy az időzítőt engedélyezni kell. Az alapértelmezett értéke false.

18.61. forráskód. Időzítés használata

```
namespace timer_1
{
    public partial class Form1 : Form
    {

        public Form1()
        {
            InitializeComponent();
            timer1.Enabled = true;
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            labelIdo.Text = DateTime.Now.ToLongTimeString();
        }
    }
}
```

18.41. feladat (Form színe időzítve – *szint: 1*). Írj programot, mely 1 másodpercenként megváltoztatja a form háttérszínét.

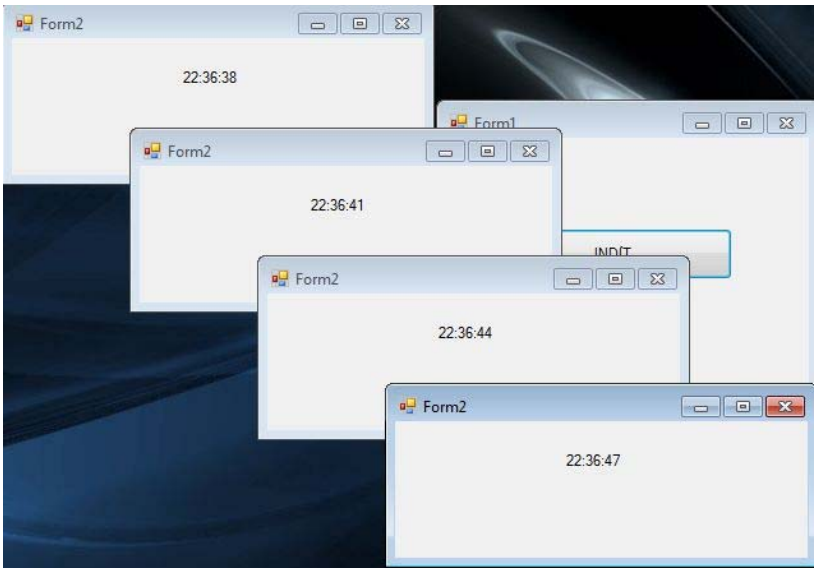
Magyarázat: Egy lehetséges megoldás a színek ciklikus változtatására a maradékos osztás használata.

18.62. forráskód. Időzítés használata

```
namespace idozit_2
{
    public partial class Form1 : Form
    {
        int ind = 0;
        Color[] szinek = new Color[5]
            { Color.White, Color.Blue, Color.Beige,
              Color.Black, Color.DarkSeaGreen };

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            timer1.Interval = 1000;
            timer1.Enabled = true;
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            this.BackColor = szinek[ind % 5];
            ind++;
        }
    }
}
```

18.42. feladat (Új formok születése és halála – *szint: 2*). Írj programot, 2 másodpercenként megjelenít egy 20 másodpercig látható formot. Ezt 1 percig csinálja. Minden új form más más feliratot tartalmazzon.

18.43. ábra. Új formok születése és halála



Magyarázat: Több timerre is szükségünk lesz mind a főformon, ami méri a 2 másodperceket, mind a létrejövő formokon, ami a 20 másodpercekért felel. A főformon az 1 percet is figyelni kell.

18.63. forráskód. A vezérlés a főformon

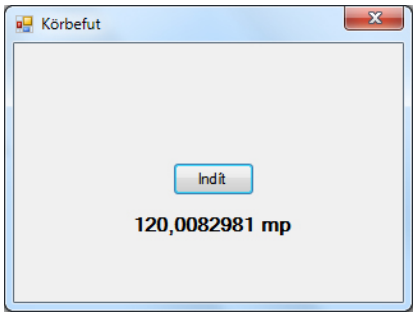
```
DateTime kezdido = new DateTime(); int x, y; TimeSpan eltelt;
private void button1_Click(object sender, EventArgs e)
{  timer1.Enabled = true;
   timer1.Interval = 2000;
   kezdido = DateTime.Now; }
private void timer1_Tick(object sender, EventArgs e)
{  eltelt = DateTime.Now - kezdido;
   if (eltelt.Seconds > 60) this.Close();
   Form2 frm = new Form2();
   frm.label1.Text = DateTime.Now.ToLongTimeString();
   frm.x = x % 800;
   frm.y = y % 600;
   frm.Show();
   x += 100;
   y += 100; } }
```

18.64. forráskód. A megjelenő formok

```
private void Form2_Load(object sender, EventArgs e)
{ timer1.Enabled = true; }
private void timer1_Tick(object sender, EventArgs e)
{ this.Close(); }
private void Form2_Shown(object sender, EventArgs e)
{ this.Top = y;
  this.Left = x; }
```

18.43. feladat (Keringő form – *szint: 2*). Írjon programot, mely egy formot mozgat a képernyőn körbe, 1 másodperces időközzel lépkedve. A mozgás 2 percig menjen az indítástól. Használj felsorolás típust a mozgás irányának tárolásához.

18.44. ábra. Körbe fut a képernyőn



18.65. forráskód. Felsorolás típus létrehozása

```
namespace Korbefut
{  enum Iranyok
   {  Jobbra,
      Le,
      Balra,
      Fel
   }
```

```
}
```

Magyarázat: Az éppen aktuális irány tárolásához hozzunk létre egy felsorolás típust (Jobbra,Le,Balra,Fel).

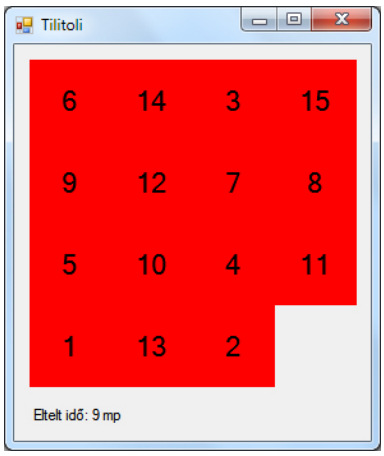
18.66. forráskód. Feladat megoldása

```
private void timer_Tick(object sender, EventArgs e)
{  label.Text = String.Format("{0} mp", stopper.Elapsed.TotalSeconds);
   if (stopper.Elapsed.TotalSeconds >= 120)
   {
      timer.Enabled = false;
      stopper.Stop();
      Left = (Screen.PrimaryScreen.Bounds.Width - Width) / 2;
      Top = (Screen.PrimaryScreen.Bounds.Height - Height) / 2;
      return;
   }
   switch (irany)
   {  case Iranyok.Jobbra:
         if (Left + lepes < Screen.PrimaryScreen.Bounds.Width - Width)
            Left += lepes;
         else
         {  Left = Screen.PrimaryScreen.Bounds.Width - Width;
            irany = Iranyok.Le;
         }  break;
      case Iranyok.Le:
         if (Top + lepes <Screen.PrimaryScreen.Bounds.Height - Height)
            Top += lepes;
         else
         {  Top = Screen.PrimaryScreen.Bounds.Height - Height;
            irany = Iranyok.Balra;
         }  break;
      case Iranyok.Balra:
         if (Left - lepes > 0)
            Left -= lepes;
         else
         {  Left = 0;
            irany = Iranyok.Fel;
         }  break;
      case Iranyok.Fel:
         if (Top - lepes > 0)
            Top -= lepes;
         else
         {  Top = 0;
            irany = Iranyok.Jobbra;
         }  break;
   }
}
```

Magyarázat: Az aktuális állapot alapján mindig tudhatjuk, hogy merre kell mozgatni a *form*-ot, és melyik lesz a következő irány, ha elértük a képernyő megfelelő szélét. Figyeljünk arra, hogy mivel fix nagyságú lépéseket teszünk, nem biztos, hogy pontosan a képernyő széléhez ér az ablak majd a mozgás végén, ezért igazítsuk be (ne lógjon ki és ne is maradjon rés).

18.44. feladat (TiliToli játék – *szint: 3*). Írjon pogramot, mely megvalósítja a TiliToli játékot. Közben valósítsa meg az időmérést is.

18.45. ábra. TiliToli játék



Magyarázat: A játék lemezkéit egy-egy dinamikusan létrehozott label komponens valósítsa meg. Az újonnan létrehozott labeleket tároljuk el egy kétdimenziós tömbben, és mindegyikéhez ugyanazt az eseménykezelőt rendeljük. A közös metódus a sender objektumban utazó label pozíciójából „találja ki”, hogy ő melyik a rácsban. Ha a lyuk melletti lemezre kattintunk, akkor az cseréljen helyet a lyukkal. Készítsünk függvényt, ami a labelek feliratából és helyes sorrendjéből meghatározza, hogy jó-e az elrendezés.

18.67. forráskód. A kérdéses metódusok

```
void ujLabel_Click(object sender, EventArgs e)
{
   if (!timer.Enabled)
   {
      timer.Enabled = true;
   }
   Label kep = (sender as Label);
   int i = kep.Left / meret;
   int j = kep.Top / meret;
   if (i > 0 && labelek[i - 1, j] == null)
   {
```

```
            Csere(i, j, i - 1, j);
        }
        else
            if (i < N - 1 && labelek[i + 1, j] == null)
            {
                Csere(i, j, i + 1, j);
            }
            else
                if (j > 0 && labelek[i, j - 1] == null)
                {
                    Csere(i, j, i, j - 1);
                }
                else
                    if (j < N - 1 && labelek[i, j + 1] == null)
                    {
                        Csere(i, j, i, j + 1);
                    }
        if (MindegyikJoHelyen())
        {
            timer.Enabled = false;
            MessageBox.Show("Gratulálok, kész! {0}",Lb_Ido.Text );
        }
    }

    private bool MindegyikJoHelyen()
    {
        bool jo = true;
        for (int db = 0; (db < N * N) && jo; db++)
        {
            int i = db % N;
            int j = db / N;
            if (labelek[i, j] != null)
            {
                jo = Convert.ToInt32(labelek[i, j].Text) == (db + 1);
            }
        }
        return jo;
    }
```

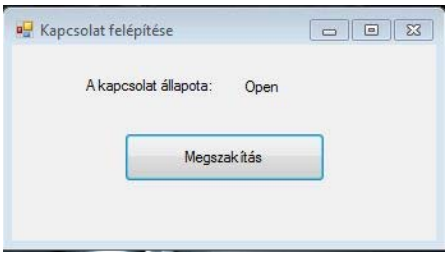
19. fejezet - Adatkezelés (szerző: Radványi Tibor)

Tartalom

SqlConnection, ConnectionString

19.1. feladat (Kapcsolat felépítése – *szint: 1*). Írjon programot, melynek a segítségével kapcsolódni tud a *Minta* adatbázishoz. Használja a szerver telepítésekor megadott *sa* felhasználót és jelszavát. Majd egy labelben jelenítse meg a kapcsolat állapotát. Tudja a kapcsolatot bontani is.

19.1. ábra. Kapcsolódás adatbázishoz



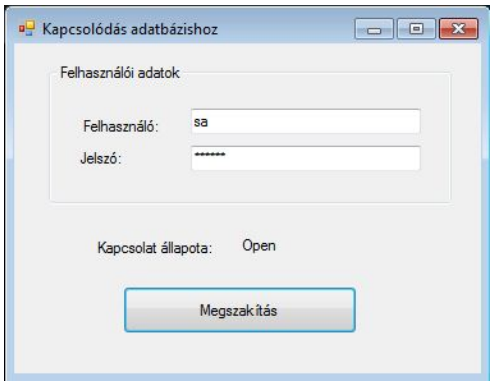
Magyarázat: Figyeljünk arra, hogy az adatkezeléshez szükségünk van a *System.Data.SqlClient* névtérre.

19.1. forráskód. Kapcsolódás adatbázishoz

```
namespace sqlconn_1
{public partial class Form1 : Form
    {private string ConnSt =
        "server=localhost\\MSSQL2005;database=Minta;uid=sa;pwd=master";
        private SqlConnection sconn ;

        public Form1()
        {  InitializeComponent();
           buttonCon.Text = "Kapcsolódás...";
           labelKapcs.Text = String.Empty;
           sconn = new SqlConnection(ConnSt); }

        private void buttonCon_Click(object sender, EventArgs e)
        {  if (sconn.State == ConnectionState.Closed)
           {
              sconn.Open();
              labelKapcs.Text = sconn.State.ToString();
              buttonCon.Text = "Megszakítás"; }
           else
           {  sconn.Close();
              labelKapcs.Text = sconn.State.ToString();
              buttonCon.Text = "Kapcsolódás"; } } }
}
```

19.2. feladat (Felhasználó azonosítás – *szint: 2*). Írjon programot, melynek a segítségével kapcsolódni tud a *Minta* adatbázishoz. A kapcsolat kiépítéséhez kérje be a felhasználó nevét és jelszavát. Egy labelben jelenítse meg a a kapcsolat állapotát. Tudja a kapcsolatot bontani is.

19.2. ábra. Kapcsolódás adatbázishoz adatbekéréssel



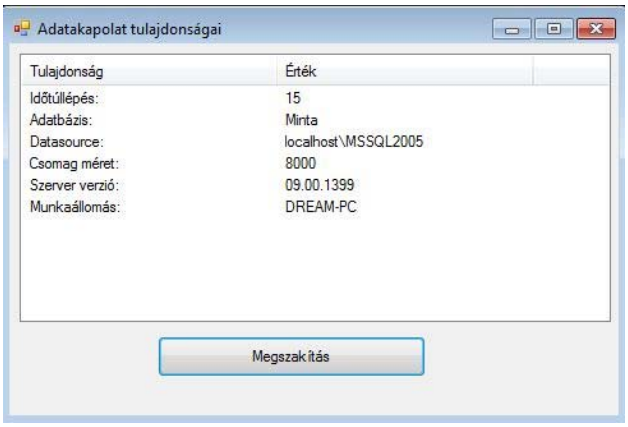
Magyarázat: A connectionString összeállítására kell figyelni.

19.2. forráskód. Kapcsolódás adatbázishoz adatbekéréssel

```
private void buttonKapcs_Click(object sender, EventArgs e)
{
   if (textBoxUser.Text != String.Empty &&
      textBoxPw.Text != String.Empty)
   {
      string CS = ConnSt + ";uid=" + textBoxUser.Text +
        ";pwd=" + textBoxPw.Text;
      if (sconn.State == ConnectionState.Closed)
      {  sconn.ConnectionString = CS;
         sconn.Open();
         labelCon.Text = sconn.State.ToString();
         buttonKapcs.Text = "Megszakítás";
      }
      else
      {  sconn.Close();
         labelCon.Text = sconn.State.ToString();
         buttonKapcs.Text = "Kapcsolódás";
      }
}
}
```

19.3. feladat (Kapcsolat jellemzői – *szint: 2*). Írjon programot, melynek a segítségével kapcsolódni tud a *Minta* adatbázishoz. Használja a szerver telepítésekor megadott *sa* felhasználót és jelszavát. Majd írassa ki a kapcsolat jellemzőit a formra: Időtúllépés, Adatbázis neve, szerver neve, csomagméret, a host gép azonosítója és az adatbázis állapota. Tudja a kapcsolatot bontani is.

19.3. ábra. A kapcsolat jellemzői



19.3. forráskód. A deklarációk

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace sqlconn_3
{
   public partial class Form1 : Form
   {
      private string ConnSt =
```

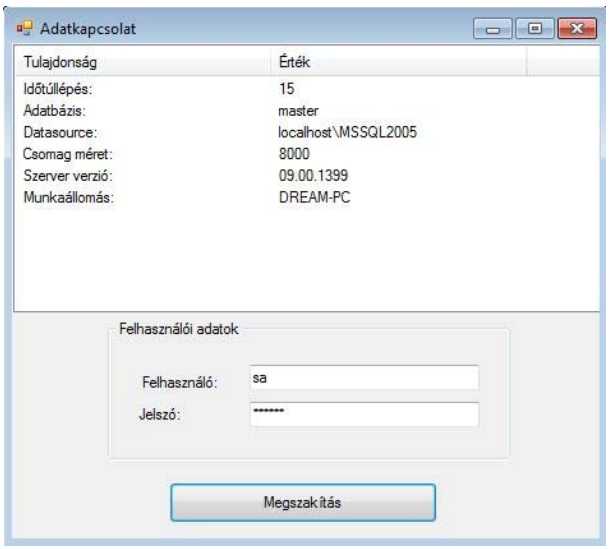
```
    "server=localhost\\MSSQL2005;database=Minta;uid=sa;pwd=master";
     private SqlConnection sconn;

     public Form1()
     {
        InitializeComponent();
        buttonCon.Text = "Kapcsolódás...";
        sconn = new SqlConnection(ConnSt);
     }
```

19.4. forráskód. A kapcsolat jellemzői

```
private void buttonCon_Click(object sender, EventArgs e)
{
   lvAdat.Items.Clear();
   if (sconn.State == ConnectionState.Closed)
   {
      sconn.Open();
      buttonCon.Text = "Megszakítás";
      ListViewItem l = new ListViewItem();
      l.Text = "Időtúllépés: ";
      l.SubItems.Add(sconn.ConnectionTimeout.ToString());
      lvAdat.Items.Add(l);
      ListViewItem ldb = new ListViewItem();
      ldb.Text = "Adatbázis: ";
      ldb.SubItems.Add(sconn.Database.ToString());
      lvAdat.Items.Add(ldb);
      ListViewItem lds = new ListViewItem();
      lds.Text = "Datasource: ";
      lds.SubItems.Add(sconn.DataSource.ToString());
      lvAdat.Items.Add(lds);
      ListViewItem lps = new ListViewItem();
      lps.Text = "Csomag méret: ";
      lps.SubItems.Add(sconn.PacketSize.ToString());
      lvAdat.Items.Add(lps);
      ListViewItem ls = new ListViewItem();
      ls.Text = "Szerver verzió: ";
      ls.SubItems.Add(sconn.ServerVersion.ToString());
      lvAdat.Items.Add(ls);
      ListViewItem lw = new ListViewItem();
      lw.Text = "Munkaállomás: ";
      lw.SubItems.Add(sconn.WorkstationId.ToString());
      lvAdat.Items.Add(lw);
   }
   else
   {
      sconn.Close();
      buttonCon.Text = "Kapcsolódás";
      lvAdat.Items.Clear();
      lvAdat.Items.Add(sconn.State.ToString());
   }
}
```

19.4. feladat (Kapcsolat string dinamikus használata – *szint: 3*). Írjon programot, mely egy szabványos konfigurációs file-ból (programneve.exe.config xml file-ból) beolvassa a ConnectionString tartalmát, bekéri a felhasználó nevet és a jelszót, ezzel kiegészíti a ConnectionStringet, majd kapcsolódni tud a *Minta* adatbázishoz. Majd írassa ki a kapcsolat jellemzőit a formra: Időtúllépés, Adatbázis neve, szerver neve és az adatbázis állapota. Tudja a kapcsolatot bontani is.

19.4. ábra. A kapcsolat jellemzői



Magyarázat: A konfigurációs file használatához a *System.Configuration* névteret használni kell. Ehhez nem elegendő a szokásos *using* sor, hanem adjuk a referenciákhoz is hozzá a fenti nevű dll-t. A konfigurációs fileba adjuk hozzá a ConnectionString szekciót, és töltsük is ki.

19.5. forráskód. A konfigurációs file

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
 <connectionStrings>
   <add name="betolt" connectionString="server=localhost\MSSQL2005"/>
 </connectionStrings>
</configuration>
```

Magyarázat: A konfigurációs fileból a *ConnectionStringet* olvassuk be a *ConfigurationManager* segítségével. Akár több változatot is tárolhatunk, amiket névvel különböztetünk meg.

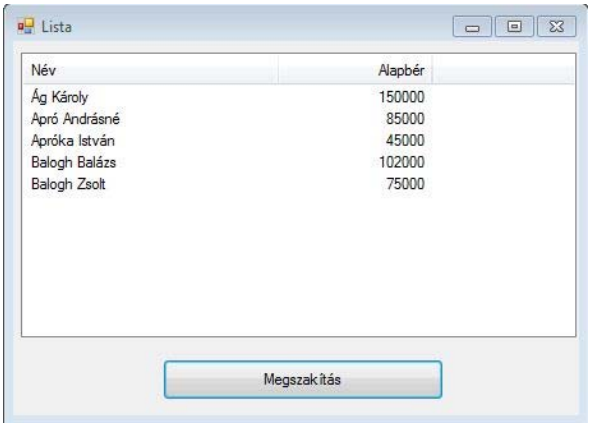
19.6. forráskód. A konfigurációs file használata

```
using System.Configuration;

   public Form1()
   { InitializeComponent();
    buttonCon.Text = "Kapcsolódás...";
    sconn = new SqlConnection();
    ConnSt =
   ConfigurationManager.ConnectionStrings["betolt"].ConnectionString; }
```

Az SqlCommand

19.5. feladat (Tábla tartalmának lekérése és megjelenítése – *szint: 2*). Írjon programot, melynek a segítségével kapcsolódni tud a *Minta* adatbázishoz. Olvassa be az Alkalmazottak táblából az első 5 nevet és fizetést, név szerint rendezve, és jelenítse meg egy *ListView*-ban. Használja az *SqlDataReader* és az *SqlCommand* osztályt.

19.5. ábra. Működés közben.

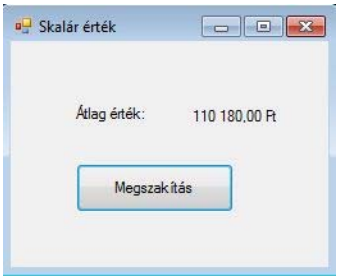


19.7. forráskód. —

```
public Form1()
{
   InitializeComponent();
   buttonCon.Text = "Kapcsolódás...";
   sconn = new SqlConnection(ConnSt);
   scomm = sconn.CreateCommand();
   scomm.CommandText = "select top 5 Nev,"
      +"Alapber from Alkalmazottak order by Nev";
}


private void Beolvas()
{
   SqlDataReader r = scomm.ExecuteReader();
   while (r.Read())
   {
      ListViewItem l = new ListViewItem();
      l.Text = (string)r[0];
      l.SubItems.Add(r[1].ToString());
      lvAdat.Items.Add(l);
   }
   r.Close();
}
```

19.6. feladat (Skalár érték lekérése és megjelenítése – *szint: 2*). Írjon programot, melynek a segítségével kapcsolódni tud a *Minta* adatbázishoz. Olvassa be az Alkalmazottak táblából az átlagfizetést és jelenítse meg egy *Label*-en. Használja az *SqlCommand* osztályt.

19.6. ábra. Működés közben.

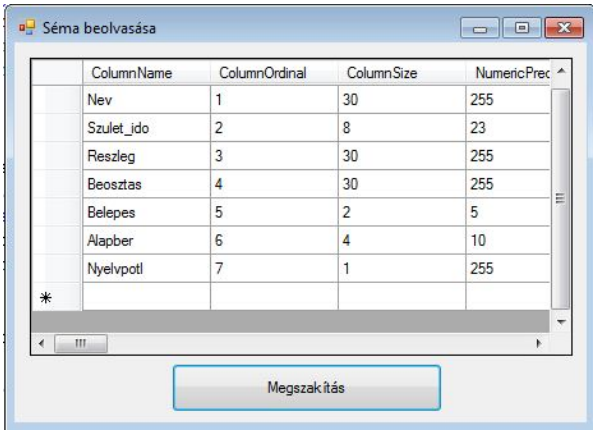


19.8. forráskód. —

```
public Form1()
{
   InitializeComponent();
   sconn = new SqlConnection(ConnSt);
   scomm = sconn.CreateCommand();
   scomm.CommandText = "select avg(alapber) from Alkalmazottak";
}

private double Beolvas()
{
   double atl;
   atl = Convert.ToDouble(scomm.ExecuteScalar());
   return atl;
}
```

19.7. feladat (Séma lekérése és megjelenítése – *szint: 2*). Írjon programot, melynek a segítségével kapcsolódni tud a *Minta* adatbázishoz. Olvassa be az Alkalmazottak tábla séma szerkezetét, és jelenítse meg. Használja az *SqlCommand* osztályt.

19.7. ábra. Működés közben.

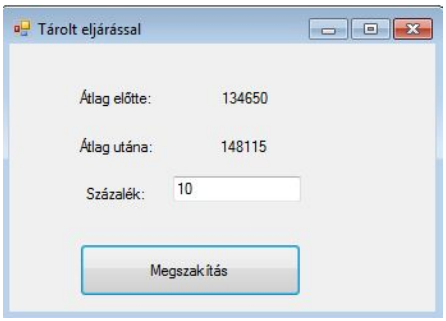


19.9. forráskód. —

```
public Form1()
{
   InitializeComponent();
   buttonCon.Text = "Kapcsolódás...";
   sconn = new SqlConnection(ConnSt);
   scomm = sconn.CreateCommand();
   scomm.CommandText = "select * from Alkalmazottak";
}

private void Beolvas()
{
   SqlDataReader r = scomm.ExecuteReader(CommandBehavior.SchemaOnly);
   DataTable dt = r.GetSchemaTable();
   dgvAdat.DataSource = dt;
}
```

19.8. feladat (Tárolt eljárás futtatása – *szint: 4*). Írjon programot, melynek a segítségével kapcsolódni tud a *Minta* adatbázishoz. Készítsen egy tárolt eljárást, mely a paraméterként kapott %-os mértékben megemeli az alkalmazottak fizetését, és kiszámolja az átlagfizetést az emelés előtt és után, és ezeket az adatokat visszaadja. Jelenítse meg az átlagértékeket. Használja az *SqlCommand* osztályt.

19.8. ábra. Működés közben.



19.10. forráskód. —

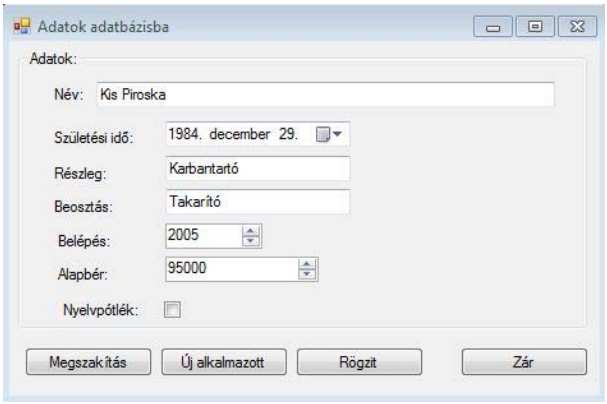
```
CREATE PROCEDURE spAtlagNovel
(
 @szazalek float,
 @elotte float output,
 @utana float output
)
AS
BEGIN
 select @elotte=avg(alapber) from Alkalmazottak

 update Alkalmazottak set alapber = (1 + @szazalek / 100) * alapber

 select @utana=avg(alapber) from Alkalmazottak
END
GO
```

19.11. forráskód. —

```
namespace sqlcomm_4
{
   public partial class Form1 : Form
   {
      private string ConnSt =
     "server=localhost\\MSSQL2005;database=Minta;uid=sa;pwd=master";
      private SqlConnection sconn;
      private SqlCommand scomm;
      private double szazalek;

      public Form1()
      {
         InitializeComponent();
         sconn = new SqlConnection(ConnSt);
         scomm = sconn.CreateCommand();
         scomm.CommandType = CommandType.StoredProcedure;
         scomm.CommandText = "spAtlagnovel";
      }

      private void buttonCon_Click(object sender, EventArgs e)
      {
         if (sconn.State == ConnectionState.Closed)
         {  sconn.Open();
            buttonCon.Text = "Megszakítás";
            if (textBox1.Text != String.Empty &&
               double.TryParse(textBox1.Text, out szazalek))
            {
               Beolvas();
            } }
         else
         {  sconn.Close();
            buttonCon.Text = "Kapcsolódás"; }
      }

      private void Beolvas()
      {
         double el, ut;
         scomm.Parameters.Add("szazalek", SqlDbType.Float);
         scomm.Parameters["szazalek"].Value = szazalek;
         scomm.Parameters.Add("elotte", SqlDbType.Float);
         scomm.Parameters["elotte"].Direction =
             ParameterDirection.Output;
         scomm.Parameters.Add("utana", SqlDbType.Float);
         scomm.Parameters["utana"].Direction =
             ParameterDirection.Output;
         if (sconn.State == ConnectionState.Closed) sconn.Open();
         scomm.ExecuteNonQuery();
         el = (double)scomm.Parameters["elotte"].Value;
         ut = (double)scomm.Parameters["utana"].Value;
         labelAtlElotte.Text = el.ToString();
         labelAtlUtana.Text = ut.ToString();
      }
   }
}
```

19.9. feladat (Adatfelvitel adatbázisba – *szint: 4*). Írjon programot, melynek a segítségével kapcsolódni tud a *Minta* adatbázishoz. Kérje be egy új alkalmazott adatait, és szúrja be az új rekordot. Használja az *SqlCommand* osztályt.

19.9. ábra. Működés közben.



19.12. forráskód. Deklarációk

```
namespace sqlcomm_5
{
   public partial class Form1 : Form
   {
      private string ConnSt =
     "server=localhost\\MSSQL2005;database=Minta;uid=sa;pwd=master";
      private SqlConnection sconn;
      private SqlCommand scomm;

      public Form1()
      {
         InitializeComponent();
         sconn = new SqlConnection(ConnSt);
         scomm = sconn.CreateCommand();
         scomm.CommandType = CommandType.Text;
         buttonCon.Text = "Kapcsolódás";
         gbAdat.Visible = false;
         buttonRogz.Enabled = false;
         buttonUj.Enabled = false;
         scomm.CommandText = "insert into Alkalmazottak " +
         " (nev, Szulet_ido, reszleg, beosztas, belepes, alapber, "+
              "nyelvpotl) values " +
         "(@nev, @Szulet_ido, @reszleg, @beosztas, @belepes,"+
            " @alapber, @nyelvpotl)"; }
```

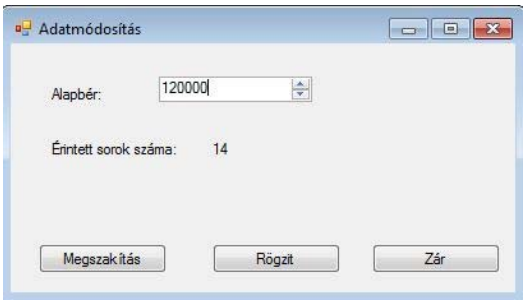
19.13. forráskód. Végrehajtás

```
public Form1()
{
   InitializeComponent();
   sconn = new SqlConnection(ConnSt);
   scomm = sconn.CreateCommand();
   scomm.CommandType = CommandType.Text;
   buttonCon.Text = "Kapcsolódás";
   gbAdat.Visible = false;
   buttonRogz.Enabled = false;
   buttonUj.Enabled = false;
   scomm.CommandText = "insert into Alkalmazottak " +
      " (nev, Szulet_ido, reszleg, beosztas, belepes,"
     +"alapber, nyelvpotl) values (@nev, @Szulet_ido,"
     +"@reszleg, @beosztas, @belepes, @alapber, @nyelvpotl)";
}

private void buttonRogz_Click(object sender, EventArgs e)
{
   if (textBoxNev.Text != String.Empty &&
      numericAlapBer.Value > 0)
   {
      scomm.Parameters.Clear();
      scomm.Parameters.Add("nev", SqlDbType.VarChar, 30);
      scomm.Parameters["nev"].Value = textBoxNev.Text;
      scomm.Parameters.Add("Szulet_ido", SqlDbType.DateTime);
      scomm.Parameters["Szulet_ido"].Value =
                          dateTimePickerSzdatum.Value;
      scomm.Parameters.Add("reszleg", SqlDbType.VarChar, 30);
      scomm.Parameters["reszleg"].Value = textBoxReszleg.Text;
      scomm.Parameters.Add("beosztas", SqlDbType.VarChar, 30);
      scomm.Parameters["beosztas"].Value = textBoxBeo.Text;
      scomm.Parameters.Add("belepes", SqlDbType.SmallInt);
      scomm.Parameters["belepes"].Value = numericBelep.Value;
      scomm.Parameters.Add("alapber", SqlDbType.Int);
      scomm.Parameters["alapber"].Value = numericAlapBer.Value;
      scomm.Parameters.Add("nyelvpotl", SqlDbType.Bit);
      scomm.Parameters["nyelvpotl"].Value =
                          checkBoxNyelvPotlek.Checked;
      try
      {
         if (sconn.State == ConnectionState.Closed) sconn.Open();
         scomm.ExecuteNonQuery();
      }
      catch (Exception ex)
      {
         MessageBox.Show("Adatrögzítés sikertelen." + ex.Message,
            "Figyelem ", MessageBoxButtons.OK, MessageBoxIcon.Error);
      }
      SetAlapHelyzet();
   }
```

```
}
```

19.10. feladat (Adatkarbantartás adatbázisban 2 – *szint: 4*). Írjon programot, melynek a segítségével kapcsolódni tud a *Minta* adatbázishoz. Módosítsa a fizetéseket egy bekért értékre, de csak a segédmunkások esetén. A művelet végrehajtása után írassa ki a művelet által érintett sorok számát a formra. Használja az *SqlCommand* osztályt.

19.10. ábra. Működés közben.



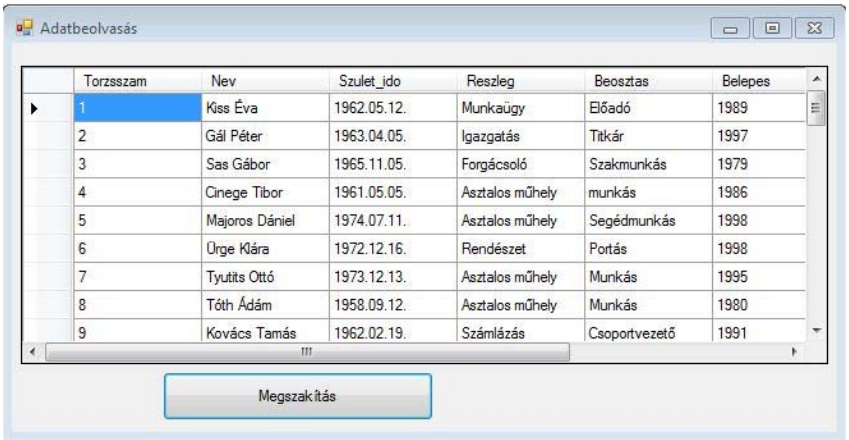
19.14. forráskód. Működés

```
public Form1()
{  InitializeComponent();
   sconn = new SqlConnection(ConnSt);
   scomm = sconn.CreateCommand();
   scomm.CommandType = CommandType.Text;
   buttonCon.Text = "Kapcsolódás";
   scomm.CommandText = "update Alkalmazottak " +
      " set alapber = @alapber where beosztas = @beosztas ";
}

private void buttonRogz_Click(object sender, EventArgs e)
{  if (numericAlapBer.Value >= 0)
   {
      scomm.Parameters.Clear();
      scomm.Parameters.Add("beosztas", SqlDbType.VarChar, 30);
      scomm.Parameters["beosztas"].Value = c_beo;
      scomm.Parameters.Add("alapber", SqlDbType.Int);
      scomm.Parameters["alapber"].Value = numericAlapBer.Value;
      try
      {
         if (sconn.State == ConnectionState.Closed) sconn.Open();
         int rows = scomm.ExecuteNonQuery();
         labelSorok.Text = rows.ToString();
      }
      catch (Exception ex)
      {
         MessageBox.Show("Adatrögzítés sikertelen." + ex.Message,
            "Figyelem ", MessageBoxButtons.OK, MessageBoxIcon.Error);
      }
      SetAlapHelyzet();
   } }
```

Adatok megjelenítése, adatkötés, DateSet és DataTable

19.11. feladat (Tábla tartalmának lekérése és megjelenítése – *szint: 2*). Írjon programot, melynek a segítségével kapcsolódni tud a *Minta* adatbázishoz. Olvassa be az Alkalmazottak táblából az adatokat, név szerint rendezve, és jelenítse meg egy *DataGridView*-ban. Használja az *DataTable* és az *DataAdapter* osztályt.

19.11. ábra. Működés közben.



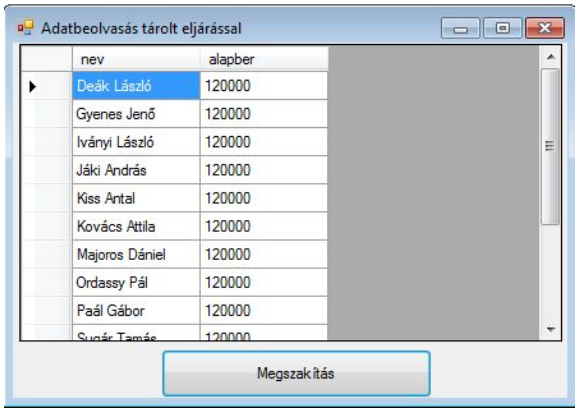
19.15. forráskód. A program mejelenése

```
private string ConnSt =
"server=localhost\\MSSQL2005;database=Minta;uid=sa;pwd=master";
```

```
   private SqlConnection sconn;
   private SqlCommand scomm;
   private DataTable dt = new DataTable();
   private SqlDataAdapter da;
   public Form1()
   {  InitializeComponent();
      buttonCon.Text = "Kapcsolódás...";
      sconn = new SqlConnection(ConnSt);
      scomm = sconn.CreateCommand();
      scomm.CommandText = "select * from Alkalmazottak"; }
   private void buttonCon_Click(object sender, EventArgs e)
   {  if (sconn.State == ConnectionState.Closed)
      {  sconn.Open();
         buttonCon.Text = "Megszakítás";
         Beolvas(); }
      else
      {  sconn.Close();
         buttonCon.Text = "Kapcsolódás"; } }
   private void Beolvas()
   {  if (sconn.State == ConnectionState.Closed) sconn.Open();
      da = new SqlDataAdapter(scomm);
      dgvAdat.DataSource = dt;
      da.Fill(dt); }
```

19.12. feladat (Tábla tartalmának lekérése tárolt eljárással – *szint: 3*). Írjon programot, melynek a segítségével kapcsolódni tud a *Minta* adatbázishoz. Olvassa be az Alkalmazottak táblából a segédmunkások adatait, név szerint rendezve, és jelenítse meg egy *DataGridView*-ban. Használja a *DataTable* és az *DataAdapter* osztályokat. A beolvasást egy tárolt eljárás végezze el!

19.12. ábra. Működés közben.

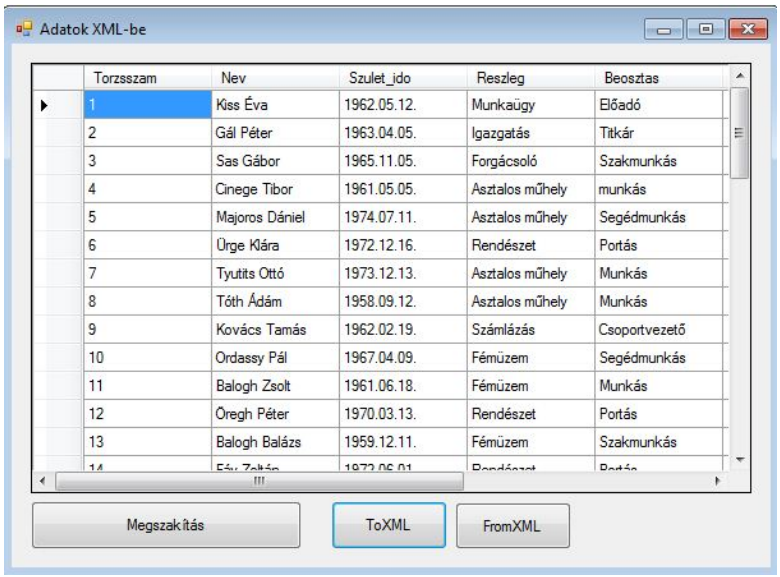


19.16. forráskód. Beolvasás

```
public Form1()
{  InitializeComponent();
   buttonCon.Text = "Kapcsolódás...";
   sconn = new SqlConnection(ConnSt);
   scomm = sconn.CreateCommand();
   DataColumn dcn = new DataColumn("nev");
   dt.Columns.Add(dcn);
   DataColumn dca = new DataColumn("alapber");
   dt.Columns.Add(dca); }
private void Beolvas()
{  if (sconn.State == ConnectionState.Closed) sconn.Open();
   scomm.CommandType = CommandType.StoredProcedure;
   scomm.CommandText = "spAdatLeker";
   scomm.Parameters.Add("beo", SqlDbType.NVarChar, 30);
   scomm.Parameters["beo"].Value = "Segédmunkás";
   SqlDataReader dr = scomm.ExecuteReader();
   while (dr.Read())
   {  DataRow r = dt.NewRow();
      r["nev"] = dr[0];
      r["alapber"] = dr[1];
      dt.Rows.Add(r); }
   dgvAdat.DataSource = dt; }
```

19.13. feladat (Táblák tartalmának lekérése és megjelenítése XML-ben – *szint: 5*). Írjon programot, melynek a segítségével kapcsolódni tud a *Minta* adatbázishoz. Olvassa be az Alkalmazottak táblából az adatokat. Jelenítse meg *DataGridView*-ban. Használja az *DataTable*, *DataSet* és az *DataAdapter* osztályt. Lehessen Az adatokat elmenteni egy tetszőlegesen kiválasztott helyre, egy megadott nevű XML-fileba. Adjon lehetőséget, hogy egy XML file-ból a mentett adatokat vissza lehessen olvasni.

19.13. ábra. Működés közben.

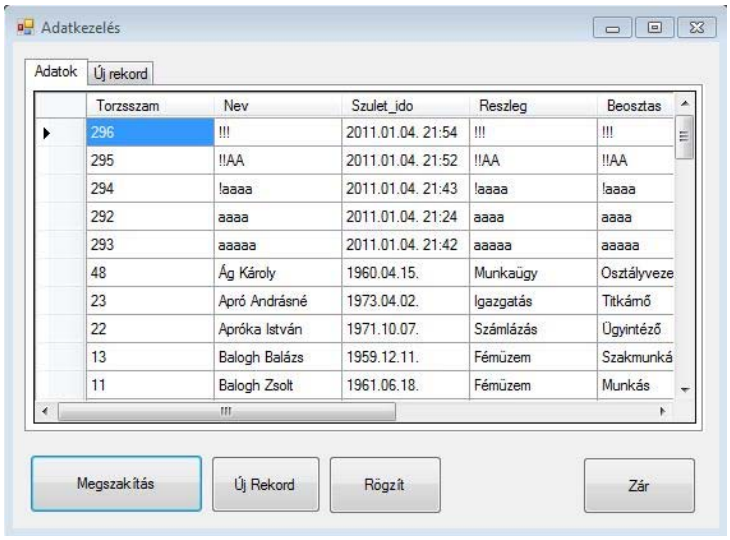


19.17. forráskód. XML használata

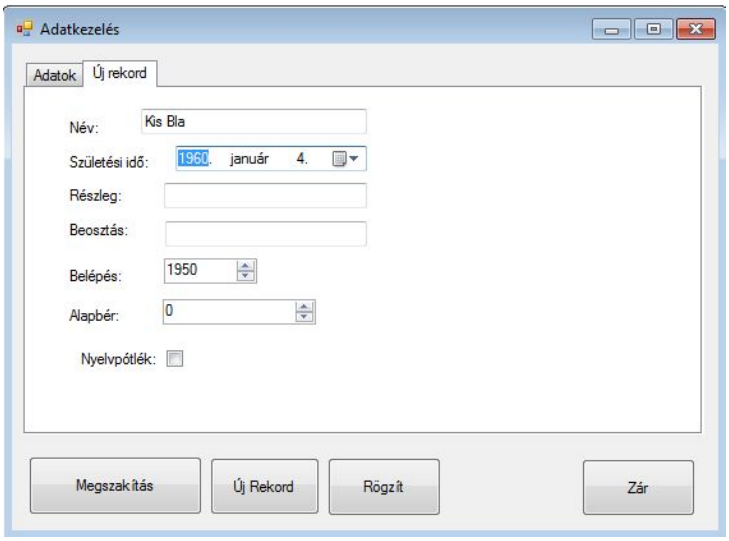
```
private void buttonToXML_Click(object sender, EventArgs e)
{
   SaveFileDialog sf = new SaveFileDialog();
   sf.DefaultExt = "xml";
   if (sf.ShowDialog() == DialogResult.OK)
   {
   ds.WriteXml(sf.FileName);
   }
}

private void buttonFromXML_Click(object sender, EventArgs e)
{
   OpenFileDialog of = new OpenFileDialog();
   of.DefaultExt = "xml";
   if (of.ShowDialog() == DialogResult.OK)
   {
      ds.ReadXml(of.FileName);
      dgvAdat.DataSource = ds.Tables[0];
   }
}
```

19.14. feladat (Adatbevitel – *szint: 3*). Írjon programot, melynek a segítségével kapcsolódni tud a *Minta* adatbázishoz. Kérjen be egy új alkalmazott adatait, és szúrja be az új rekordot.

19.14. ábra. Működés közben.



19.15. ábra. Működés közben.



19.18. forráskód. Deklaráció és kapcsolódás

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace adatkotes_4
{
   public partial class Form1 : Form
   {
      private string ConnSt =
     "server=localhost\\MSSQL2005;database=Minta;uid=sa;pwd=master";
      private SqlConnection sconn;
      private SqlCommand scomm;
      private DataTable dt = new DataTable();
      private SqlDataAdapter da;

      public Form1()
      {
         InitializeComponent();
         buttonCon.Text = "Kapcsolódás...";
         sconn = new SqlConnection(ConnSt);
         scomm = sconn.CreateCommand();
         buttonRogzit.Enabled = false;
         buttonUj.Enabled = false;
      }

      private void buttonCon_Click(object sender, EventArgs e)
      {
         if (sconn.State == ConnectionState.Closed)
         {
            sconn.Open();
            buttonCon.Text = "Megszakítás";
            buttonRogzit.Enabled = true;
            buttonUj.Enabled = true;
            Beolvas();
         }
         else
         {
            sconn.Close();
            buttonCon.Text = "Kapcsolódás";
            buttonRogzit.Enabled = false;
            buttonUj.Enabled = false;
         }
      }
```

19.19. forráskód. Rögzítés

```
private void Beolvas()
{
   if (sconn.State == ConnectionState.Closed) sconn.Open();
   scomm.CommandType = CommandType.Text;
   scomm.CommandText = "select * from Alkalmazottak order by nev";
   da = new SqlDataAdapter(scomm);
   dgvAdat.DataSource = dt;
   da.Fill(dt);
}

private void buttonZar_Click(object sender, EventArgs e)
{
   Close();
}

private void buttonUj_Click(object sender, EventArgs e)
{
   tabControl1.SelectTab(1);
```

```
}

private void buttonRogzit_Click(object sender, EventArgs e)
{
   string ins = "insert into Alkalmazottak values "+
      "(@nev, @szulet_ido, @reszleg, @beosztas, "+
      "@belepes, @alapber, @nyelvpotl )";
   SqlCommand sc = new SqlCommand(ins, sconn);
   sc.Parameters.Add("nev", SqlDbType.NVarChar, 30);
   sc.Parameters["nev"].Value = textBoxNev.Text;
   sc.Parameters.Add("reszleg", SqlDbType.NVarChar, 30);
   sc.Parameters["reszleg"].Value = textBoxNev.Text;
   sc.Parameters.Add("beosztas", SqlDbType.NVarChar, 30);
   sc.Parameters["beosztas"].Value = textBoxNev.Text;
   sc.Parameters.Add("belepes", SqlDbType.Int);
   sc.Parameters["belepes"].Value = numericBelep.Value;
   sc.Parameters.Add("alapber", SqlDbType.Int);
   sc.Parameters["alapber"].Value = numAlapBer.Value;
   sc.Parameters.Add("szulet_ido", SqlDbType.DateTime);
   sc.Parameters["szulet_ido"].Value = dateTimePicker1.Value;
   sc.Parameters.Add("nyelvpotl", SqlDbType.Bit);
   sc.Parameters["nyelvpotl"].Value = cbNyelvP.Checked;
   if (sconn.State == ConnectionState.Closed) sconn.Open();
   try
   {
      sc.ExecuteNonQuery();
   }
   catch (Exception ex)
   {
      MessageBox.Show("Hiba az adatrögzítésben. " + ex.Message,
         "Hiba", MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
   dgvAdat.DataSource = null;
   dt.Columns.Clear();
   dt.Rows.Clear();
   Beolvas();
   dgvAdat.DataSource = dt;
   tabControl1.SelectTab(0);
}
}
}
```

Tárolt eljárások írása és használata

19.15. feladat (Átmeneti tábla lekérdezéssel – *szint: 2*). Írjon tárolt eljárást, mely az *Alkalmazottak* minta táblából kiválogatja a szakmunkások nevét és alapbérét, és ezt egy szaki nevű táblába tárolja el, és végül kilistázza.

19.20. forráskód. Átmeneti tábla

```
CREATE PROCEDURE [dbo].[spTarolt1]
AS
BEGIN
      select nev, alapber into szaki
            from Alkalmazottak
            where beosztas = 'Szakmunkás'
      select * from szaki order by nev
END
```

19.16. feladat (Automatikus kulcsérték visszakérése – *szint: 2*). Írjon tárolt eljárást, mely az *Alkalmazottak* minta táblába beszúr egy új rekordot, majd visszaadja az új rekord elsődleges kulcs értékét. Az elsődleges kulcs érték automatikusan generálódik.

19.21. forráskód. @@Identity

```
CREATE PROCEDURE [dbo].[spTarolt2]
(
@azon int output
)
AS
BEGIN
      insert into Alkalmazottak (nev, alapber, nyelvpotl)
            values ('Kiss Ádám', 145000, 1)

      SELECT @azon = @@Identity;

END
```

Magyarázat: Figyeljen arra, hogy hogyan lehet a tárolt eljárást futtani, helyes paraméter használattal.

19.22. forráskód. Paraméteres futtatás

```
declare @i int
exec spTarolt2 @azon = @i output
select i = @i
```

19.17. feladat (Kurzor használata – *szint: 5*). Írjon tárolt eljárást, mely az *Alkalmazottak* minta táblában megemeli az alkalmazottak fizetését. Ha egy alkalmazott az átlagnál kevesebbet keres, akkor 20%-al, míg ha többet, akkor 10%-al. A feladat megoldásához használjon kurzort.

Magyarázat: A kurzor deklarációja és használata egyszerű. Használat után ne felejtse el bezárni és felszabadítani. Tartsa szem előtt, hogy a kurzor használata lassú folyamat!

19.23. forráskód. Kurzor használata

```
CREATE PROCEDURE [dbo].[spTarolt3]
AS
```

```
BEGIN
declare @atl float
declare @alapber float
declare @tsz int
declare @emel float
declare cur_alk cursor for
      select alapber, torzsszam from Alkalmazottak
select @atl = avg(alapber) from Alkalmazottak
open cur_alk;
fetch next from cur_alk into @alapber, @tsz
while @@fetch_status = 0
      begin
            if @alapber < @atl set @emel = 20
                  else set @emel = 10
            update Alkalmazottak
                  set alapber = (1 + @emel / 100) * alapber
                  where torzsszam = @tsz
            fetch next from cur_alk into @alapber, @tsz
      end
close cur_alk;
deallocate cur_alk;
END
```

19.18. feladat (Vezérlési szerkezetek 1 – *szint: 2*). Írjon tárolt eljárást, amely kiírja a Fibonacci-sorozat első 10 elemét (0,1,1,2,3,...).

19.24. forráskód. Vezérlési szerkezet

```
CREATE PROCEDURE [dbo].[spTarolt4]
AS
BEGIN
declare @fib1 int
declare @fib2 int
declare @ujfib int
declare @i int
set @fib1 = 0
set @fib2 = 1
set @i = 1
print @fib1
print @fib2
while @i < 9
      begin
      set @i = @i + 1
      set @ujfib = @fib1 + @fib2
      print @ujfib
      set @fib1 = @fib2
      set @fib2 = @ujfib
      end
END
```

19.19. feladat (Legnagyobb különbség – *szint: 2*). Írjon tárolt eljárást, mely a felhasználó által megadott részlegnél kiszámítja a legnagyobb fizetési különbséget.

19.25. forráskód. Egy ötlet bejárás nélkül

```
CREATE PROCEDURE [dbo].[spTarolt5]
(@reszleg nvarchar(30))
AS
BEGIN
select max(alapber) - min(alapber) from Alkalmazottak
where reszleg = @reszleg
END
```

Magyarázat: Figyeljen a paraméteres futtatásra

19.26. forráskód. Paraméteres futtatás

```
exec spTarolt5 @reszleg = 'Rendészet'
```

20. fejezet - Grafikai feladatok (szerző: Kovács Emőd)

Tartalom

Grafikai feladatok

20.1. feladat (Kör rajzolása – *szint: 4*).

Készítsünk programot, mely az alább ismertetett MidpointKor és Korpontok metódusok felhasználásával köröket rajzol a képernyőre. A rajzolt kör minden esetben az egérrel való kattintás helyén jelenjen meg 50 pixelnyi sugárral.

Magyarázat: Az egér koordinátái a MouseEventArgs e változóból olvasható ki a MouseUp form eseményben. A 20.12 forrásszövegben láthatjuk, hogy a Bx és a By statikus változók. (static int Bx,By;)

A rajzolást a Form1_Paint eseményben kell meghívni. A MouseUp eseményben a Refresh hívással aktivizálhatjuk a Form1_Paint metódust. Ez törli az előzőleg kirajzolt felületet, majd kirajzolja az új kört.

A feladat továbbfejlesztéseként alakítsuk át a programot úgy, hogy az egérrel megadhatjuk a kör átmérőjét (lásd: 20.23).

20.2. feladat (Bezier görbe – *szint: 4*). Készítsünk WinForm programot, amely egy tetszőlegesen változtatható formájú Bezier görbét rajzol a képernyőre!

Magyarázat: Deklaráljuk, és kezdőértéket adunk a később felhasználandó segédváltozóknak. A bool típusú nyom változó, azt figyeli, hogy lenyomtuk-e már az egér gombját, azaz megadtuk-e már, hogy hol legyen a görbénk első pontja. Ha ezt nem tennénk meg, akkor a Form-unkra kirajzolódna egy vonal, ami a (0,0) koordinátából indul ki, és a kattintásunk helye a végpontjának a koordinátái.

A következő változó, amelynek a neve max, a Form-ra kitehető maximális pontok számát jelenti. Aztán egy Point típusú mezők tárolására alkalmas tömböt hozunk létre. Most kell felhasználnunk a max változót, hisz meg kell adnunk, hogy milyen hossza legyen a tömbnek.

Felmerülhet a kérdés, hogy miért nem használunk ArrayList-et, mivel akkor nem kellene a max változó sem, és az Add metódusa Object típust vár, tehát Point-ot is tehetnénk bele. Ez igaz, de, mikor az ArrayList-ben lévő elemeknek értéket próbálnánk adni (konkrétan a MouseMove eseményben) a fordító hibát jelezne a típuskényszerítés miatt.

Majd jön az n változó, ebben a Form-ra kirakott pontok számát tartjuk nyilván, ez kezdetben 0. A mozgat a kontroll poligon pontjainak mozgatásához szükséges, illetve még szükségünk van egy Graphics típusú változóra is, deklarációját ezért raktuk ide, és nem a Paint eseménybe (lásd: 20.34 forrásszöveg).

A programot elindítva, a vezérlők (a két nyomógomb, illetve két checkbox) megjelennek, meghívódik a Paint esemény, g itt megkapja az értékét, de az n értéke még mindig 0 (lásd: 20.45) Ezután, ha kattintunk, két esemény is meghívódik egymás után: a MouseDown(lenyomtuk az egér gombját), és a MouseUp(felengedtük)

Ez a MouseDown esemény törzse. Az egérgomb lenyomása után, három dologra kell ügyelni. Az első, hogy kattintottunk-e egy pontra törlési céllal, illetve kattintás után nyomva tartjuk-e a bal gombot, mivel ekkor pontot át akarjuk helyezni, vagy kattintottunk-e egy helyre a Form-on, és ide új pontot szeretnénk felvenni

Először is megvizsgáljuk, hogy a checkBox2 be van-e jelölve. Ha igen, akkor törölni akarunk.

Egy while-ciklussal végigmegyünk a Pontok tömbön, és megnézzük, hogy melyik pontra kattintottak. Ha megtaláltuk, meghívjuk a TorolPont metódust, átadva azt az indexet, ahol megtaláltuk a pontot. A mozgat változónak mindezek mellett adnunk kell valamilyen értéket, hogy az ne legyen -1, és ezáltal ne lépjünk be a pontlétrehozó feltételbe.

A TorolPont metódus működésére még visszatérünk, egyelőre folytassuk a munkát az esemény vizsgálatával. Ha tehát nincs bejelölve a checkBox2, de mégis valamelyik pontra kattintottunk, akkor nagy a valószínűsége, hogy ezt mozgatni akarjuk. Mindössze annyi a dolgunk, hogy megkeressük azt a pontot, amire kattintottak, és az indexét értékül adjuk a mozgat változónak, amelyet aztán felhasználunk a MouseMove eseményben. Ha a fentiek közül egyik sem teljesült, akkor egy új pontot akarunk létrehozni. Megnézzük rakhatunk-e még le pontot, ha igen akkor n értékét (amiben a pontokat számoljuk) növeljük eggyel, és mivel ezen indexű helyen a tömbünkben még nincs érték, arra a helyre beállítjuk az X, és Y koordinátákat, úgy, hogy értékül adjuk nekik az egér azon pozícióját, mellyel bal gombjának lenyomásakor rendelkezett.

Mivel már lenyomták az egér gombját, a nyom-ot true-ra állítjuk, és meghívjuk a Refresh-t, ami a Paint eseményt fogja újra végrehajtani. Az n értéke nagyobb, mint 0, így a Gorbe, és a PontKi metódusok végrehajtódnak. Mielőtt ezekre rátérnénk, térjünk vissza a TorolPont metódusra, és MouseMove eseményre. Kezdjük a TorolPont-al .

A metódus paraméterében megkapja azt az indexet, ahol a törlendő pont van (ez az i). Majd következik egy ciklus, ami ettől az i-től kezdve végigmegy a tömb elemein, és minden egyes értékhez az azt következőt rendeli, vagyis a törlendőt fölülírtuk az azt következővel, az azt következőt, az őt követővel, és így tovább. Az n értékét csökkenteni kell, mivel pontok száma csökkent, majd, ha még maradt pontunk, az új értékekkel újrarajzoljuk a görbét, ehhez ismét kell egy Refresh, hogy lássuk a változásokat.

A MouseMove eseményben megvizsgáljuk, hogy a mozgat értéke megváltozott-e. Ha igen, akkor pontosan azaz index az értéke, amely pontot mozgatni akarunk. Ezen index által meghatározott tömbérték X, és Y koordinátáit beállítjuk az egér aktuális pozíciójára, frissítünk, és újrarajzoljuk a görbét, hogy menet közben lássuk a változásokat.

Van még egy egérművelettel kapcsolatos esemény, amiről eddig nem beszéltünk, ez a MouseUp (lásd: 20.1 forrásszöveg).

A metódus a mozgat értékét visszaállítja 1-re (alaphelyzetbe), mivel lehetséges, hogy az egérgomb lenyomása során, (ha mozgattunk, vagy töröltünk) az értéke megváltozott. Mielőtt nekifutnánk a Gorbe metódusnak, nézzük meg, hogyan működik a törlés. A törlést egy gomb vezérli (aminek neve torlesButton). Ha erre a gombra kattintunk akkor az egy Click eseményt hoz létre.

Elindul a TorolPont eljárás, az n értékét 0-ra állítja (ha törlünk mindent, nem marad több pont). A 20.2 forrásszövegben nézzük meg azt is, hogyan néz ki a TorolPont metódus, amely létrehoz egy új Rectangle-t, ami olyan széles, és magas, mint a Form, és a 0,0 koordinátából indul ki, és ezt a Rectangle-t kifeszíti a képernyőre. Az eljárással igazság szerint nem a Form felületén lévő objektumokat töröltük, hanem a háttérszínnel lefestettük a felületét. A kontroll polinom vonalainak, és a pontoknak a rajzolását a Vonal, és a PontKi eljárással valósítjuk meg. Mindkettő egyszerűen egy helyben deklarált Pen segítségével a g grafikus metódusait használja.

A Vonal rajzolásához a Drawline paramétereként meg kell adnunk az előbb létrehozott Pen-t, valamint, hogy honnan hova akarjuk a vonalat kirajzolni, azaz két pont x, és y koordinátáját. Ezeket a koordinátákat a Pontok i-edik elemének X és Y metódusainak meghívásával kapjuk (lásd: 20.3 forrásszöveg).

A PontKi a paraméterként kapott i-edik pontot fogja kirajzolni a Drawrectangle metódus révén,ami egy négyszöget rajzol a képernyőre. Paraméterben meg kell adnunk, hogy mely x, és y koordinátára akarjuk kirajzoltatni, valamint, hogy milyen széles (Width), és magas (Height) legyen. Esetünkben 4-4 pixel.

A görbe kirajzolását (lásd: 20.4) a Gorbe metódus végzi. Minden alkalommal, amikor pontot rajzolunk ki, vagy törlünk ezt hívjuk meg a háttérben. A metódus a Bezi segédmetódussal, ami a matematikai háttere a görbe rajzolásának, számoltat ki egy pontot.

A 20.5 alapján először két pontra lesz szükségünk ezeket „ide”, és „oda” névvel illetjük, és a szakasz első és utolsó végpontját reprezentálják. Szükség van továbbá egy double típusú változóra (ez tartalmazza a lépésközöket), és természetesen a Pen eszközre a rajzoláshoz. A segédváltozóval a 0-tól indulunk, és amíg el nem érjük az 1-et 0.01-es lépésközökkel vonalakat rajzoltatunk a Drawline metódussal. Ezek a vonalak az „ide” ponttól az „oda” pontig tartanak, ha a „nyom” globális változó értéke igaz, azaz már van kirakva pont. Az „ide” minden kezdésnél az előző végpont lesz („oda”), míg az „oda” értékét a Bezi segédmetódus számolja ki. Ha az „i” értéke elérte az egyet, az azt jelenti, hogy már majdnem elértük a görbével a végpontot (megközelítettük). Ekkor rajzolunk egy vonalat, de most úgy, hogy a kezdőpont az eddig meghúzott vonal végpontja (ahogy ezt eddig is tettük), de a végpont a Pontok tömb utolsó eleme lesz. Ez a részlet garantálja, hogy a vonal pontosan a kezdőpontból a végpontig tartson. Ha a checkBox1 be van jelölve, vagyis kontroll polinomot kell rajzolni, és persze már „nyom” változó értéke true vagyis van kirakva pont, akkor meghívja a Vonal eljárást, majd a Pontok tömb összes elemét (pontokat) a PontKi eljárással kirajzolja.

A matematikai hátteret a Bezier algoritmus szolgáltatja. A Bezi segédmetódus ezt valósítja meg. Visszatérési értéke egy Pont típusú változó. Egy double paramétert vár (lásd 20.6, 20.7, 20.8 forrásszövegek), és meghívja a Bez_Suly metódust a paraméterként kapott értékkel.

20.3. feladat (Szakasz lehatárolása – *szint: 4*). Készítsük el a klasszikus szakasz lehatároló programot, mely hasonlóan működik, mint a legtöbb rajzoló program kivágás (cut) művelete. A lehatárolást az egér kattintás hatására végezzük el a kijelölés mentén.

Magyarázat:

A feladat megoldásához segítséget találunk a 20.9 forráskódban. A pixelek kirajzolásának módját a 20.10, a szakasz lehatárolását a 20.11, a midpoint szakaszok rajzolását a 20.13 forráskódban találjuk meg.

A 20.3 feladat szövege alapján, a Form1_Paint eseményben rajzoljunk egy tetszőleges vonalat a MidPoint metódus segítségével (piros színnel), majd tároljuk el koordinátákat. Rajzoljunk egy rectangle-t tetszőleges C# metódus segítségével, zöld színnel úgy, hogy az lehetőleg fedésen legyen a szakasszal! (pl.: drawRectangle) Fontos kérdések a program futásával kapcsolatban: A vágás sikeres-e? - Milyen módon ábrázolja a program a vágás eredményét? A kérdések megválaszolását a tisztelt Olvasóra bízzuk.

20.4. feladat (A DDA szakaszrajzoló – *szint: 4*). Készítsük el az ismert DDA szakaszrajzoló algoritmus programját. A DDA metódus két pont közé rajzol vonalat. A két pont x és y koordinátáit paraméterben kapja, valamit egy PaintEventArgs és egy Color típusú változót.

Magyarázat: A DDA pontokból, azaz pixelekből rajzolja ki a vonalat (lásd: 20.15).

A KoordintaRendszer a paraméterben kapott színnel DDA segítségével rajzol egy kis méretű koordináta-rendszert beosztásokkal együtt (lásd: 20.16). A Diagram metódus a DDA-val rajzolja ki a diagramokat, azaz a diagram oszlopának 3 vonalát. Az oszlop bal felső sarkának x, és y koordinátáját paraméterben kapja, valamint azt is, hogy milyen színnel rajzoljon. A 20.17 forrásszöveg alapján a Form Paint metódusában hívjuk meg az előbb említett függvényeket, így kapunk egy koordináta-rendszert, valamint a paraméterezésnek megfelelő oszlopdiagramo(ka)t, ahogy ezt a 20.18 forrásszövegben láthatjuk.

20.5. feladat (Képek átméretezése – *szint: 3*). Készítsünk a mindennapi gyakorlatban is jól használható programot, amely JPG formátumú képek csoportos átméretezését valósítja meg egységes formátum alapján (lásd: 20.19).

Magyarázat: A 20.20 forrásszövegben a fix méretre történő átméretezést láthatjuk a maradék részek kitöltésével. Amennyiben arányosítva szeretnénk átméretezni a képeket, használhatjuk a 20.21 forrásszövegben található metódust. A méretezni kívánt képek helyét meg kell adni, amely eljáráshoz a 20.22 forrásszövegben látható programrészlet vehetjük alapul. A kép megnyitása a 20.24, a konvertálás a 20.25, valamint a 20.26 forrásszövegekben látható módon történhet. A konvertálás befejeztével ne felejtsük el felszabadítani az erőforrásokat, amelyeket a program felhasznált a futása során (lásd: 20.27).

20.6. feladat (Hermit görbe rajzolása – *szint: 4*). Készítsünk olyan ablakos alkalmazást, amely egy Hermit görbét rajzol a képernyőre.

Magyarázat: Szükségünk lesz számos változóra a program készítése során. Ahogy azt a 20.28 forrásszövegben láthatjuk, a maxp konstans a maximális pontok számát tartalmazza. A „pontmozg" az éppen mozgatott pontot tárolja. A szak2 egy pontokból álló tömb, amely maxp + 1 darab pontot tartalmaz. Az aktp az éppen aktuális pont sorszáma. Kell még 8 pont a segédvonalak, érintő, és a görbe kirajzolásához, valamint deklarálnunk kell egy Graphics típusú változót. A megoldáshoz használjuk a 20.28 forrásszövegben található Bezier algoritmusunkat is. A 20.29 forrásszövegben a hermithatarok metódus paraméterének át kell adnunk két int-et, ami ezekhez számolja ki és állítja be a hozzájuk tartozó Point típusú érintőket. A paraméterben kapott számok a szak Point típusú tömb valahányadik elemének a számai. A 20.30 forrásszövegben található hermiteu3 metódus paraméterben vár 4 pontot, és egy double típusú változót, és az ismert matematikai képlet alapján számolja ki egy pont koordinátáit, amit vissza is ad. A 20.31 forrásszövegben bemutatott erintorajz 4 ponthoz rajzol érintőt. A pontokat paraméterben kapja. A hermitrajz (lásd: 20.32) paraméterében azt a színt kell megadni, amellyel ki akarjuk rajzolni a görbét. A polirajz2 poliszin paraméterként kapott színnel dolgozik. Ha bejelöltük, akkor meghívja a hermitrajz függvényt, ami a Hermit görbét rajzolja ki. Paraméterként a fekete színt adja át neki. Illetve ha be van jelölve akkor a bezier görbét is kirajzolja a matematikai képlet alapján (lásd 20.33, és 20.35 forrásszövegek). A pontjelolo paraméterében kapott Point típusú elemekből álló tömb összes elemét rajzolja ki a pontaszin színnel, valamint erintoszin színnel a szak tömb szomszédos pontjait összekötő szakaszokat. A pontjel a paraméterben kapott tömb paraméterként kapott helyen álló elemét (egy pontot) rajzol ki. A paraméter nélküli poliujra függvény a képernyőtörlés után újrarajzolja a polinomot, valamit visszaállítja a pontok alapértékeit. A segedvonal függvény a polinomunkhoz rajzolja ki a segédvonalakat (lásd: 20.39, 20.37, 20.38). Ez a metódus paraméterként kapja a szak Point tömböt, és egy int-et, amely megmutatja, hogy hányadik ponthoz rajzoljon vonalat, valamint egy színt, hogy a pontot milyen színnel rajzolja ki (lásd: 20.36).

A rajz alapjainak elkészítését a paraméter nélküli initrajz függvény végzi (lásd: 20.41). Meghívni a Form indításakor kell és előre deklarált változókat állít be. A Form Paint eseménye az előre deklarált Graphics típusú változónknak ad értéket, majd meghívja a pontjelölő2 függvényt,paraméterként átadja a szak Point-okat tartalmazó tömböt, így az kirajzolhatja a pontokat, majd meghívja a polirajz függvényt, paraméterben egy színnel, ami a polinom színe (lásd: 20.42, 20.43, és 20.44 forrásszövegek). Ha az egér gombját felengedjük a pontmozg változónk érékét -1-re állítjuk, és újrarajzoltatjuk a polinomot a poliujra metódussal (lásd: 20.46 forrásszöveg). A további Form-on lévő objektumokhoz rendelt metódusokat a 20.47 forrásszövegben találjuk. A button1 megnyomásakor a program befejezi a futását, a Form1_Load esemény meghívja az initrajz függvényt, azaz elkészíti a görbe alapjait. Ha a checkBox2 megváltozik frissítünk, és a numericUpDown

20.7. feladat (Ellipszis rajzolása – *szint: 3*). Készítsünk olyan rajzoló programot, amely az egérrel kijelölt befoglaló keretbe ellipsziseket rajzol. A rajzoláshoz használhatjuk a C# beépített metódusát, vagy készíthetünk egyet magunk is.

20.8. feladat (Rajzprogram – *szint: 3*). Készítsünk a Paint alkalmazás mintájára rajzoló programot, amelyben az egér segítségével lehet rajzolni, szakaszokat húzni pontok között, valamint területeket színezni.

A fejezet forráskódjai 1.

20.1. forráskód.

```
private void Form1_MouseUp(object sender, MouseEventArgs e)
{
   mozgat = -1;
}
```

20.2. forráskód.

```
private void TorolPont()
{
   Rectangle rect = new Rectangle(this.ClientRectangle.Left,
                          this.ClientRectangle.Top,
                          this.ClientRectangle.Width,
                          this.ClientRectangle.Height);
                          this.RectangleToScreen(rect);
}
```

20.3. forráskód.

```
private void Vonal()
{
   Pen p = new Pen(Color.Black);
   for (int i = 1; i < n; i++)
         g.DrawLine(p, Pontok[i].X,
                   Pontok[i].Y,
                   Pontok[i + 1].X,
                   Pontok[i + 1].Y);
}
```

20.4. forráskód.

```
private void PontKi(int i)
{
   Pen p = new Pen(Color.Blue);
   try
   {
   g.DrawRectangle(p,
       Pontok[i].X - 2,
       Pontok[i].Y - 2, 4, 4);
   }
}
```

20.5. forráskód.

```
private void Gorbe()
{
   Point ide, oda;
   double i = 0;
   Pen p = new Pen(Color.Red);
   oda = Pontok[1];
   while (i <= 1)
   {
      ide = oda;
      oda = Bezi(i);
         if (nyom == true)
            g.DrawLine(p, ide.X, ide.Y, oda.X, oda.Y);
      i += 0.01;
   }
   ide=oda;
   oda=Pontok[n];
   g.DrawLine(p, ide.X, ide.Y, oda.X, oda.Y);
   if(checkBox1.Checked&&nyom==true)
      Vonal();
   for(int j=1;j<n;j++)
      PontKi(j);
}
```

20.6. forráskód.

```
private Point Bezi(double t)
{
   double x = 0, y = 0, b = 0;
   for (int i = 0; i < n; i++)
   {
      b = Bez_Suly(i, n - 1, t);
      x = (x + Pontok[i + 1].X * b);
      y = (y + Pontok[i + 1].Y * b);
   }
   return new Point((int)x, (int)y);
}
```

20.7. forráskód.

```
private double Bez_Suly(int i, int n, double t)
{
   double temp = N_alatt_I(n, i);
   for (int j = 1; j <= i; j++) temp *= t;
   for (int j = 1; j <= n - 1; j++) temp *= (1 - t);
   return temp;
}
```

20.8. forráskód.

```
private int Faktor(int n)
{
   int tmp = 1;
   for (int i = 2; i <= n; i++) tmp *= i;
   return tmp;
}

private int N_alatt_I(int n, int i)
{
   return Faktor(n) / (Faktor(i) * Faktor(n - i));
}
```

20.9. forráskód.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
```

```
using System.Windows.Forms;
using System.Data;

namespace Lehatarolas
{
      /// <summary>
      /// Summary description for Form1.
      /// </summary>
      public class Form1 : System.Windows.Forms.Form
      {
            private System.ComponentModel.IContainer components;

            public Form1()
            {
                  //
                  // Required for Windows Form Designer support
                  //
                  InitializeComponent();

                  //
                  // TODO: Add any constructor code after InitializeComponent call
                  //
            }

            /// <summary>
            /// Clean up any resources being used.
            /// </summary>
            protected override void Dispose( bool disposing )
            {
                  if( disposing )
                  {
                        if (components != null)
                        {
                              components.Dispose();
                        }
                  }
                  base.Dispose( disposing );
            }

            #region Windows Form Designer generated code
            /// <summary>
            /// Required method for Designer support - do not modify
            /// the contents of this method with the code editor.
            /// </summary>
      ///
            private void InitializeComponent()
            {
                  this.components = new System.ComponentModel.Container();
                  this.timer1 = new System.Windows.Forms.Timer(this.components);
                  this.mainMenu1 = new System.Windows.Forms.MainMenu();
                  this.menuItem1 = new System.Windows.Forms.MenuItem();
                  this.menuItem2 = new System.Windows.Forms.MenuItem();
                  this.menuItem3 = new System.Windows.Forms.MenuItem();
                  this.menuItem4 = new System.Windows.Forms.MenuItem();
                  //
                  // timer1
                  //
                  this.timer1.Interval = 1000;
                  this.timer1.Tick += new System.EventHandler(this.timer1_Tick);
                  //
                  // mainMenu1
                  //
                  this.mainMenu1.MenuItems.AddRange(new System.Windows.Forms.MenuItem[] {
                                                                                                                        this.me
                  //
                  // menuItem1
                  //
                  this.menuItem1.Index = 0;
                  this.menuItem1.MenuItems.AddRange(new System.Windows.Forms.MenuItem[] {
                                                                                                                        this.me
                                                                                                                        this.me
                                                                                                                        this.me
                  this.menuItem1.Text = "Line";
                  //
                  // menuItem2
                  //
                  this.menuItem2.Index = 0;
                  this.menuItem2.Text = "Alap";
                  this.menuItem2.Click += new System.EventHandler(this.menuItem2_Click);
                  //
                  // menuItem3
                  //
                  this.menuItem3.Index = 1;
                  this.menuItem3.Text = "DDA";
                  this.menuItem3.Click += new System.EventHandler(this.menuItem3_Click);
                  //
                  // menuItem4
                  //
                  this.menuItem4.Index = 2;
                  this.menuItem4.Text = "Mid-Point";
                  this.menuItem4.Click += new System.EventHandler(this.menuItem4_Click);
                  //
                  // Form1
                  //
                  this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
                  this.ClientSize = new System.Drawing.Size(440, 366);
                  this.Menu = this.mainMenu1;
                  this.Name = "Form1";
                  this.Text = "Cohen-Sutherland";
                  this.MouseDown += new System.Windows.Forms.MouseEventHandler(this.Form1_MouseDown);
                  this.MouseUp += new System.Windows.Forms.MouseEventHandler(this.Form1_MouseUp);
                  this.Paint += new System.Windows.Forms.PaintEventHandler(this.Form1_Paint);
                  this.MouseMove += new System.Windows.Forms.MouseEventHandler(this.Form1_MouseMove);
            }
            #endregion
```

```
Point kezd = new Point(0,0);
Point vege = new Point(0,0);
Point l1 = new Point(0,0);
Point l2 = new Point(0,0);
int mit = 1;
bool line = false;
private System.Windows.Forms.MainMenu mainMenu1;
private System.Windows.Forms.MenuItem menuItem1;
private System.Windows.Forms.MenuItem menuItem2;
private System.Windows.Forms.MenuItem menuItem3;
private System.Windows.Forms.MenuItem menuItem4;
private System.Windows.Forms.Timer timer1;

[STAThread]
static void Main()
{
      Application.Run(new Form1());
}

private void Form1_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e)
{
      timer1.Enabled = false;
      line = false;
      kezd = new Point(e.X,e.Y);
}

private void Form1_MouseUp(object sender, System.Windows.Forms.MouseEventArgs e)
{
      vege = new Point(e.X,e.Y);
      line = true;
      timer1.Enabled = true;
      this.Refresh();
}

private void Form1_MouseMove(object sender, System.Windows.Forms.MouseEventArgs e)
{
      if(e.Button == MouseButtons.Left)
      {
            vege = new Point(e.X,e.Y);
            this.Refresh();
      }
}

private void timer1_Tick(object sender, System.EventArgs e)
{
      this.Refresh();
}

private void Form1_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{
      timer1.Enabled = false;
      Random r = new Random();
       e . Graphics . DrawRectangle ( Pens . Black , kezd . X , kezd . Y , vege . X  - kezd . X , vege . Y  - kezd . Y );
       if ( line )
       {
             l1 . X   =   r . Next ( this . Width );
             l1 . Y   =   r . Next ( this . Height );
             l2 . X   =   r . Next ( this . Width );
             l2 . Y   =   r . Next ( this . Height );
             switch ( mit )
             {
                   case   1   :
                         e . Graphics . DrawLine ( Pens . Green , l1 , l2 );
                         break ;
                   case   2   :
                         DDA ( e . Graphics , Color . Green , l1 . X , l1 . Y , l2 . X , l2 . Y );
                         break ;
                   case   3   :
                         MidPoint ( e . Graphics , Color . Green , l1 . X , l1 . Y , l2 . X , l2 . Y );
                         break ;
             }
             CohenSutherlandLineClipAndDraw (
                   l1 . X , l1 . Y , l2 . X , l2 . Y ,
                   kezd . X , vege . X , kezd . Y , vege . Y ,
                   e . Graphics , Color . Red );
       }
       timer1 . Enabled   =   true ;
 }

 //   Nincs   PutPixel   C #- ban ,   csak   Bitmap   pontjait   érjük   el .
 private   void   drawPixel ( Graphics   g , Color   color , int   x , int   y )
 {
       using   ( Bitmap   b   =   new   Bitmap (1,1))
       {
             b . SetPixel (0,0, color );
             g . DrawImageUnscaled ( b , x , y );
       }
 }
 //****   szakasz   rajzoló   metódusok   kezdő -   és   végponttal
 //***   az   utolsó   pixelt   nem   nem   éri   el
 public   void   DrawLine ( Pen ,   Point ,   Point );
 public   void   DrawLine ( Pen ,   PointF ,   PointF );
 public   void   DrawLine ( Pen ,   int ,   int ,   int ,   int );
 public   void   DrawLine ( Pen ,   float ,   float ,   float ,   float );

 # region   DDA
 //*************************************************************
 //*************************** DDA *****************************
 //*************************************************************
 private   void   DDA ( Graphics   g , Color   color , int   x1 , int   y1 , int   x2 , int   y2 )
 {
       int   dx   =   x2  - x1 ;
       int   dy   =   y2  - y1 ;
       float   xn , yn ;
```

```
      float   x   =   x1 ;
      float   y   =   y1 ;
      float   hossz   =   Math . Abs ( dx );
      if ( hossz < Math . Abs ( dy ))   hossz   =   Math . Abs ( dy );
      xn   =   dx   /   hossz ;
      yn   =   dy   /   hossz ;
      for ( int   i =1; i < Convert . ToInt32 ( hossz ); i ++)
      {
            drawPixel ( g , color , Convert . ToInt32 ( x ), Convert . ToInt32 ( y ));
            x   +=   xn ;
            y   +=   yn ;
      }
}
# endregion

# region   Mid  - Point
//************************************************************
//******************** Mid  - Point *********************************
//************************************************************
private   void   MidPoint ( Graphics   g , Color   color , int   x1 , int   y1 , int   x2 , int   y2 )
{
      int   dx , dy ;
      int   x = x1 , y = y1 ;
      dx = x2  - x1 ;
      dy = y2  - y1 ;
      if   ( dx * dy ==0)
      {
            if   ( dx ==0)
            {
                  y =( y1 < y2 )? y1 : y2 ;
                  y2 =( y1 < y2 )? y2 : y1 ;
                  for   (;   y <= y2 ;   y ++)
                        drawPixel ( g , color , x1 , y );
            }
            else
            {
                  x =( x1 < x2 )? x1 : x2 ;
                  x2 =( x1 < x2 )? x2 : x1 ;
                  for   (;   x <= x2 ;   x ++)
                        drawPixel ( g , color , x , y1 );
            }
      }
      else
      {
            float   m ;
            int   c1 , c2 , p ;
            int   bx =1, by =1;
            m = dy /( float ) dx ;
            dx =( dx >0)? dx :- dx ;
            dy =( dy >0)? dy :- dy ;
            c1 = dy + dy ;
            c2 =( dy  - dx )<<1;
            p =( dy + dy )- dx ;
            drawPixel ( g , color , x , y );
            if   ( m >=-1   &&   m <=1)
            {
                  if   ( x2 < x1 )
                  {
                        bx =-1;
                        if   ( y2 < y1 )
                              by =-1;
                        while   ( x > x2 )
      {
                              x += bx ;
                              if   ( p >=0)
                              {
                                    p += c2 ;
                                    y += by ;
                              }
                              else
                                    p += c1 ;
                              drawPixel ( g , color , x , y );
                        }
                  }
                  else
                  {
                        if   ( y2 < y1 )
                              by =-1;
                        while   ( x < x2 )
                        {
                              x += bx ;
                              if   ( p >=0)
                              {
                                    p += c2 ;
                                    y += by ;
                              }
                              else
                                    p += c1 ;
                              drawPixel ( g , color , x , y );
                        }
                  }
            }
            else
            {
                  c1 = dx + dx ;
                  c2 =( dx  - dy )<<1;
                  p =( dx + dx )- dy ;
                  drawPixel ( g , color , x , y );
                  if   ( y2 < y1 )
                  {
                        by =-1;
                        if   ( x2 < x1 )
                              bx =-1;
                        while   ( y > y2 )
                        {
```

```
                                    y += by ;
                                    if   ( p >=0)
                                    {
                                          p += c2 ;
                                          x += bx ;
                                    }
                                    else
                                          p += c1 ;
                                    drawPixel ( g , color , x , y );
                              }
                        }
                        else
                        {
                              if   ( x2 < x1 )
                              bx =-1;
                              while   ( y < y2 )
                              {
                                    y += by ;
                                    if   ( p >=0)
                                    {
                                          p += c2 ;
                                          x += bx ;
                                    }
                                    else
                                          p += c1 ;
                                    drawPixel ( g , color , x , y );
                              }
                        }
                  }
            }
      }
      # endregion

      # region   Cohen  - Sutherland
      //************************************************************
      //************************ Cohen  - Sutherland *********************
      //************************************************************
      uint   TOP =0 x1 ;
      uint   BOTTOM =0 x2 ;
      uint   RIGHT =0 x4 ;
      uint   LEFT =0 x8 ;

      private   void   CohenSutherlandLineClipAndDraw (
            double   x0 ,   double   y0 ,   double   x1 ,   double   y1 ,   double   xmin ,   double   xmax ,
            double   ymin ,   double   ymax ,   Graphics   g ,   Color   c )
      {
            uint   outcode0 ,   outcode1 ,   outcodeOut ;
            bool   accept = false ,   done = false ;

            outcode0   =   CompOutCode (   x0 ,   y0 ,   xmin ,   ymin ,   xmax ,   ymax   );
            outcode1   =   CompOutCode (   x1 ,   y1 ,   xmin ,   ymin ,   xmax ,   ymax   );
            do
            {
                  if (   0==( outcode0 | outcode1 )   )// egyenes   az   ablakon   belul   van ( siman   kirajzolni )
                  {
                        accept = true ;   done = true ;
                  }
                  else   if (   0!=( outcode0 & outcode1 ))// egyenes   nem   erinti   az   ablakot ( kihagyni )
                        done   =   true ;
                  else
                  {
                        double   x ,   y ;
                        if (   outcode0 !=0   )
                              outcodeOut   =   outcode0 ;
                        else   outcodeOut   =   outcode1 ;

                        if   (   0!=( outcodeOut   &   TOP )   )
                        {
                              x   =   x0   +   ( x1  - x0 )*( ymax    -   y0 )/( y1    -   y0 );
                              y   =   ymax ;
                        }
                        else   if   (0!=( outcodeOut   &   BOTTOM ))
                        {
                              x   =   x0   +   ( x1  - x0 )*( ymin    -   y0 )/( y1    -   y0 );
                              y   =   ymin ;
                        }
                        else   if   (0!=( outcodeOut   &   RIGHT ))
                        {
                              y   =   y0   +   ( y1  - y0 )*( xmax    -   x0 )/( x1    -   x0 );
                              x   =   xmax ;
                        }
                        else
                        {
                              y   =   y0   +   ( y1  - y0 )*( xmin    -   x0 )/( x1    -   x0 );
                              x   =   xmin ;
                        }
                        if   (   outcodeOut   ==   outcode0   )
                        {
                              x0   =   x ;
                              y0   =   y ;
                              outcode0   =   CompOutCode (   x0 ,   y0 ,   xmin , ymin , xmax , ymax );
                        }
                        else
                        {
                              x1   =   x ;
                              y1   =   y ;
                              outcode1   =   CompOutCode (   x1 ,   y1 ,   xmin , ymin , xmax , ymax );
                        }
                  }
            }
      while (   done == false   );

            if   (   accept   )
            {
                  MidPoint ( g , c ,( int ) x0 ,( int ) y0 ,( int ) x1 ,( int ) y1 );
```

```
                }
            }

            private   uint   CompOutCode   (
                  double   x ,   double   y ,
                  double   xmin ,   double   ymin ,
                  double   xmax ,   double   ymax   )
            {
                  uint   code =0;
                  if   (   y   >   ymax   )
                        code   |=   TOP ;
                  else   if (   y   <   ymin   )
                        code   |=   BOTTOM ;
                  if   (   x   >   xmax   )
                        code   |=   RIGHT ;
                  else   if   (   x   <   xmin   )
                        code   |=   LEFT ;

                  return   code ;
            }
            # endregion

            private   void   menuItem2_Click ( object   sender ,   System . EventArgs   e )
            {
                  mit   =   1;
            }

            private   void   menuItem3_Click ( object   sender ,   System . EventArgs   e )
            {
                  mit   =   2;
            }

            private   void   menuItem4_Click ( object   sender ,   System . EventArgs   e )
            {
                  mit   =   3;
            }
      }
}
```

20.10. forráskód.

```
drawPixel()
```

20.11. forráskód.

```
CohenSutherlandLineClipAndDraw()
```

A fejezet forráskódjai 2.

20.12. forráskód.

```
Bx=e.X;
By=e.Y;
```

20.13. forráskód.

```
MidPoint()
```

20.14. forráskód.

```
Graphics G=e.Graphics;
```

20.15. forráskód.

```
public void dda(int x1, int y1,int x2, int y2,
            PaintEventArgs p, Color c)
{
   Graphics Gf = p.Graphics;
   Bitmap Pixel = new Bitmap(1,1);
   Pixel.SetPixel(0, 0, c);
   int hossz;
   float x, y, xn, yn;
   hossz = Math.Abs(x2-x1);
   if (hossz < Math.Abs(y2 - y1))
   {
      hossz = Math.Abs(y2-y1);
   }
   xn = (float)(x2 - x1) / hossz;
   yn = (float)(y2 - y1) / hossz;
   x = x1;
   y = y1;
   for (int i = 1; i < hossz + 1; i++)
   {
      Gf.DrawImage(Pixel, x, y);
      x = x + xn;
      y = y + yn;
   }
}
```

20.16. forráskód.

```
private void KoordinataRendszer(object sender, PaintEventArgs e, Color c)
{
   Size formsize = this.ClientSize;
   Graphics g = e.Graphics;
   dda(ClientRectangle.Left+20, ClientRectangle.Top+10,ClientRectangle.Left+20,
      ClientRectangle.Bottom-10, e, c);
   dda(ClientRectangle.Left+10, ClientRectangle.Bottom-20, ClientRectangle.Right-10,
      ClientRectangle.Bottom-20, e, c);
   for (int i = 1; i < (ClientRectangle.Width / 10) - 2; i++)
   {
      dda(ClientRectangle.Left + 20 + 20 * i, ClientRectangle.Bottom - 25,
         ClientRectangle.Left + 20 + 20 * i, ClientRectangle.Bottom - 15, e, c);
   }
   for (int i = 1; i < (ClientRectangle.Height / 10) - 3; i++)
   {
      dda(ClientRectangle.Left + 15, ClientRectangle.Bottom - 20 - 20 * i,
         ClientRectangle.Left + 25, ClientRectangle.Bottom - 20 - 20 * i, e, c);
   }
}
```

20.17. forráskód.

```
private void Diagramm(int x, int y, object sender, PaintEventArgs e, Color c)
{
   Graphics Gf = e.Graphics;
   dda(x, ClientRectangle.Bottom - 20, x, ClientRectangle.Bottom - y, e, c);
   dda(x + 20, ClientRectangle.Bottom - 20, x + 20, ClientRectangle.Bottom - y, e, c);
   dda(x, ClientRectangle.Bottom - 20, x, ClientRectangle.Bottom - y, e, c);
   SolidBrush brush = new SolidBrush(c);
   Gf.FillRectangle(brush, x, y + ClientRectangle.Bottom - 20, 20, y);
}
```

20.18. forráskód.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
   Graphics g = e.Graphics;
   KoordinataRendszer(sender, e, Color.Black);
   int szazalek = 40;
   Diagramm(100, (ClientRectangle.Height / 100)*szazalek + 20, sender, e, Color.Red);
   Diagramm(200, (ClientRectangle.Height / 100)*(100 - szazalek) + 20, sender, e, Color.Blue);
   Diagramm(300, (ClientRectangle.Height / 100) * 100 + 20, sender, e, Color.Yellow);
}
```

20.19. forráskód.

```
static int Bx, By, Ex, Ey;
static double r;
static bool nyom = false;

private void MidPoint(Graphics g, Color color, int x1, int y1, int x2, int y2)
{
   int dx, dy;
   int x = x1, y = y1;
   dx = x2 - x1;
   dy = y2 - y1;
   if (dx * dy == 0)
   {
      if (dx == 0)
      {
         y = (y1 < y2) ? y1 : y2;
         y2 = (y1 < y2) ? y2 : y1;
         for (; y <= y2; y++)
                  drawPixel(g, color, x1, y);
      }
      else
      {
         x = (x1 < x2) ? x1 : x2;
         x2 = (x1 < x2) ? x2 : x1;
         for (; x <= x2; x++)
            drawPixel(g, color, x, y1);
      }
   }
   else
   {
      float m;
      int c1, c2, p;
      int bx = 1, by = 1;
      m = dy / (float)dx;
      dx = (dx > 0) ? dx : -dx;
      dy = (dy > 0) ? dy : -dy;
      c1 = dy + dy;
      c2 = (dy - dx) << 1;
      p = (dy + dy) - dx;
      drawPixel(g, color, x, y);
      if (m >= -1 && m <= 1)
      {
         if (x2 < x1)
         {
            bx = -1;
            if (y2 < y1)
               by = -1;
            while (x > x2)
            {
               x += bx;
               if (p >= 0)
```

```
                {
                   p += c2;
                   y += by;
                }
                else
                   p += c1;
                drawPixel(g, color, x, y);
             }
          }
          else
          {
             if (y2 < y1)
                by = -1;
             while (x < x2)
             {
                x += bx;
                if (p >= 0)
                {
                   p += c2;
                   y += by;
                }
                else
                   p += c1;
                drawPixel(g, color, x, y);
             }
          }
       }
       else
       {
          c1 = dx + dx;
          c2 = (dx - dy) << 1;
          p = (dx + dx) - dy;
          drawPixel(g, color, x, y);
          if (y2 < y1)
          {
             by = -1;
             if (x2 < x1)
                bx = -1;
             while (y > y2)
             {
                y += by;
                if (p >= 0)
                {
                   p += c2;
                   x += bx;
                }
                else
                   p += c1;
                drawPixel(g, color, x, y);
             }
          }
          else
          {
             if (x2 < x1)
                bx = -1;
             while (y < y2)
             {
                y += by;
                if (p >= 0)
                {
                   p += c2;
                   x += bx;
                }
                else
                   p += c1;
                drawPixel(g, color, x, y);
             }
          }
       }
    }
}


private void Korpontok(int cx, int cy, int x, int y, System.Drawing.Color szin, System.Windows.Forms.PaintEventArgs e)
{
   Graphics g = e.Graphics;
   Bitmap bm = new Bitmap(1, 1);
   bm.SetPixel(0, 0, szin);
   g.DrawImageUnscaled(bm, (int)x + cx, (int)y + cy);
   g.DrawImageUnscaled(bm, (int)y + cx, (int)x + cy);
   g.DrawImageUnscaled(bm, (int)y + cx, (int)-x + cy);
   g.DrawImageUnscaled(bm, (int)x + cx, (int)-y + cy);
   g.DrawImageUnscaled(bm, (int)-x + cx, (int)-y + cy);
   g.DrawImageUnscaled(bm, (int)-y + cx, (int)-x + cy);
   g.DrawImageUnscaled(bm, (int)-x + cx, (int)y + cy);
   g.DrawImageUnscaled(bm, (int)-y + cx, (int)x + cy);
}


private void MidpointKor(int cx, int cy, int r, System.Windows.Forms.PaintEventArgs e)
{
   int d, x, y;
   x = 0;
   y = r;
   d = 1 - r;
   Korpontok(cx, cy, x, y, Color.Red, e);
   while (y > x)
   {
      if (d < 0)
      {
         d += 2 * x + 3;
         x++;
      }
      else
      {
```

```
            d += 2 * (x - y) + 5;
            x++;
            y--;
         }
         Korpontok(cx, cy, x, y, Color.Red, e);
      }
   }
```

A fejezet forráskódjai 3.

20.20. forráskód.

```
static Image FixedSize(Image imgPhoto, int Width, int Height)
{
   int sourceWidth = imgPhoto.Width;
   int sourceHeight = imgPhoto.Height;
   int sourceX = 0;
   int sourceY = 0;
   int destX = 0;
   int destY = 0;
   float nPercent = 0;
   float nPercentW = 0;
   float nPercentH = 0;

   nPercentW = ((float)Width / (float)sourceWidth);
   nPercentH = ((float)Height / (float)sourceHeight);

   if (nPercentH < nPercentW)
   {
      nPercent = nPercentH;
      destX = System.Convert.ToInt16((Width - (sourceWidth * nPercent)) / 2);
   }
   else
   {
      nPercent = nPercentW;
      destY = System.Convert.ToInt16((Height - (sourceHeight * nPercent)) / 2);
   }

   int destWidth = (int)(sourceWidth * nPercent);
   int destHeight = (int)(sourceHeight * nPercent);

   Bitmap bmPhoto = new Bitmap(Width, Height, PixelFormat.Format24bppRgb);
   bmPhoto.SetResolution(imgPhoto.HorizontalResolution, imgPhoto.VerticalResolution);
   Graphics grPhoto = Graphics.FromImage(bmPhoto);
   Color CL = Color.FromArgb(62, 79, 108);
   grPhoto.Clear(CL);
   grPhoto.InterpolationMode = InterpolationMode.HighQualityBicubic;
   grPhoto.DrawImage(imgPhoto,
       new System.Drawing.Rectangle(destX, destY, destWidth, destHeight),
       new System.Drawing.Rectangle(sourceX, sourceY, sourceWidth, sourceHeight), GraphicsUnit.Pixel);
   grPhoto.Dispose();
   return bmPhoto;
}
```

20.21. forráskód.

```
static Image ScaleByPercent(Image imgPhoto, int Percent)
{
   float nPercent = ((float)Percent / 100);
   int sourceWidth = imgPhoto.Width;
   int sourceHeight = imgPhoto.Height;
   int sourceX = 0;
   int sourceY = 0;
   int destX = 0;
   int destY = 0;
   int destWidth = (int)(sourceWidth * nPercent);
   int destHeight = (int)(sourceHeight * nPercent);
   Bitmap bmPhoto = new Bitmap(destWidth, destHeight, PixelFormat.Format24bppRgb);
   bmPhoto.SetResolution(imgPhoto.HorizontalResolution, imgPhoto.VerticalResolution);
   Graphics grPhoto = Graphics.FromImage(bmPhoto);
   grPhoto.InterpolationMode = InterpolationMode.HighQualityBicubic;
   grPhoto.DrawImage(imgPhoto, new System.Drawing.Rectangle(destX, destY, destWidth, destHeight),
   new System.Drawing.Rectangle(sourceX, sourceY, sourceWidth, sourceHeight), GraphicsUnit.Pixel);
   grPhoto.Dispose();
   return bmPhoto;
}
```

20.22. forráskód.

```
private void Megnyitas()
{
      try
      {
            System.IO.DirectoryInfo di = new System.IO.DirectoryInfo(@textBox1.Text);
            System.IO.DirectoryInfo[] diArr = di.GetDirectories();

            foreach (System.IO.DirectoryInfo dri in diArr)
            {
                  System.IO.FileInfo[] fiArr = dri.GetFiles("*.jpg");
                  foreach (System.IO.FileInfo fri in fiArr)
                  {
```

20.23. forráskód.

```
private void Korpontok(int cx, int cy, int x, int y,
            System.Drawing.Color szin,
             System.Windows.Forms.PaintEventArgs e)
{
```

```
   Graphics g = e.Graphics;
 Bitmap bm = new Bitmap(1, 1);
 bm.SetPixel(0, 0, Color.Red);
 g.DrawImageUnscaled(bm, (int)x + cx, (int)y + cy);
 g.DrawImageUnscaled(bm, (int)y + cx, (int)x + cy);
 g.DrawImageUnscaled(bm, (int)y + cx, (int)-x + cy);
 g.DrawImageUnscaled(bm, (int)x + cx, (int)-y + cy);
 g.DrawImageUnscaled(bm, (int)-x + cx, (int)-y + cy);
 g.DrawImageUnscaled(bm, (int)-y + cx, (int)-x + cy);
 g.DrawImageUnscaled(bm, (int)-x + cx, (int)y + cy);
 g.DrawImageUnscaled(bm, (int)-y + cx, (int)x + cy);
}

private void MidpointKor(int cx, int cy, int r,
            System.Windows.Forms.PaintEventArgs e)
{
   int d, x, y;
   x = 0;
   y = r;
   d = 1 - r;
   Korpontok(cx, cy, x, y, Color.Black, e);
   while (y > x)
   {
      if (d < 0){
         d += 2 * x + 3;
         x++;
      }
      else{
         d += 2 * (x - y) + 5;
         x++;
         y--;
      }
      Korpontok(cx, cy, x, y, Color.Black, e);
   }
}
```

20.24. forráskód.

```
Image imgPhotoVert =
  Image.FromFile(@textBox1.Text +
   + dri.Name +
      + fri.Name.ToString());
```

20.25. forráskód.

```
Image imgPhoto = null;
```

20.26. forráskód.

```
if ((int)comboBox1.SelectedIndex == 0)
      imgPhoto = ScaleByPercent(imgPhotoVert, Convert.ToInt32(this.textBox3.Text));
else
      imgPhoto = FixedSize(imgPhotoVert, Convert.ToInt32(this.textBox4.Text),
            Convert.ToInt32(this.textBox5.Text));
      imgPhoto.Save(@textBox1.Text + "\\" + dri.Name + "\\" + textBox2.Text + fri.Name.ToString(),
            ImageFormat.Jpeg);
```

20.27. forráskód.

```
imgPhoto.Dispose();
}
 ...
      catch
      {
            MessageBox.Show("A konyvtar nem letezik!", "hiba",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
      }

      MessageBox.Show("Kesz!", "Atmeretezes " +
      comboBox1.SelectedItem.ToString(), MessageBoxButtons.OK,
      MessageBoxIcon.Information);
}
```

20.28. forráskód.

```
private const int maxp = 10;
private int osztas = 100;
private byte pontdb = 4;
private Color poliaszin = Color.Yellow;
private Color pontaszin = Color.Aqua;
private Color pontpszin = Color.Silver;
private Color erintopszin = Color.Silver;
private Color erintoaszin = Color.Aqua;
private int pontmozg = -1;
private double erszor = 0.5;
private Point[] szak = new Point[maxp + 1];
private double gd;
private int gm;
private Point a0, a1, a2, a3;
private Point ep3 = new Point();
private Point p3 = new Point();
private double pontossag = 40;
private int aktp;
```

```
private int j;
private Point erinto0 = new Point();
private Point erinto1 = new Point();
private Graphics g;
```

A fejezet forráskódjai 4.

20.29. forráskód.

```
private void ClearDevice()
{
   Rectangle rect = new Rectangle(this.ClientRectangle.Left, this.ClientRectangle.Top,
      this.ClientRectangle.Width, this.ClientRectangle.Height);
   this.RectangleToScreen(rect);
}
```

20.30. forráskód.

```
private void HermitHatarok(int ind1, int ind2)
{
   if (ind1 == 0)
   {
      erinto0.X = Convert.ToInt32(szak[1].X -
                  0.5 * (szak[2].X - szak[1].X));
      erinto0.Y = Convert.ToInt32(szak[1].Y -
                  0.5 * (szak[2].Y - szak[1].Y));
      erinto0.X = Convert.ToInt32(
                 (erinto0.X - szak[0].X) * erszor);
      erinto0.Y = Convert.ToInt32(
                 (erinto0.Y - szak[0].Y) * erszor);
   }
   else
   {
      erinto0.X = Convert.ToInt32((int)
               (szak[ind1 + 1].X - szak[ind1 - 1].X) * erszor);
      erinto0.Y = Convert.ToInt32(
               (szak[ind1 + 1].Y - szak[ind1 - 1].Y) * erszor);
   }
   if (ind2 == pontdb)
   {
      erinto1.X = Convert.ToInt32(szak[ind2 - 2].X +
              1.5 * (szak[ind2 - 1].X - szak[ind2 - 2].X));
      erinto1.Y = Convert.ToInt32(szak[ind2 - 2].Y +
              1.5 * (szak[ind2 - 1].Y - szak[ind2 - 2].Y));
      erinto1.X = Convert.ToInt32(
               (szak[ind2].X - erinto1.X) * erszor);
      erinto1.Y = Convert.ToInt32(
               (szak[ind2].Y - erinto1.Y) * erszor);
   }
   else
   {
      erinto1.X = Convert.ToInt32(
                 (szak[ind2 + 1].X - szak[ind1].X) * erszor);
      erinto1.Y = Convert.ToInt32(
                 (szak[ind2 + 1].Y - szak[ind1].Y) * erszor);
   }
   if (this.checkBox3.Checked == true && ind2 == 0)
   {
      erinto0.X = Convert.ToInt32((szak[0].X -
                         szak[pontdb - 1].X) * erszor);
      erinto0.Y = Convert.ToInt32((szak[0].Y -
                        szak[pontdb - 1].Y) * erszor);
      erinto1.X = Convert.ToInt32(szak[1].X - 0.5 *
                         (szak[2].X - szak[1].X));
      erinto1.Y = Convert.ToInt32(szak[1].Y - 0.5 *
                         (szak[2].Y - szak[1].Y));
      erinto1.X = Convert.ToInt32((erinto1.X -
                          szak[0].X) * erszor);
      erinto1.Y = Convert.ToInt32((erinto1.Y -
                          szak[1].Y) * erszor);
   }
}
```

20.31. forráskód.

```
private Point hermiteu3(Point p0, Point p1, Point t0,
                          Point t1, double u)
{
   Point pont = new Point();
   a0 = p0;
   a1 = t0;
   a2.X = 3 * (p1.X - p0.X) - 2 * t0.X - t1.X;
   a2.Y = 3 * (p1.Y - p0.Y) - 2 * t0.Y - t1.Y;
   a3.X = -2 * (p1.X - p0.X) + t0.X + t1.X;
   a3.Y = -2 * (p1.Y - p0.Y) + t0.Y + t1.Y;
   pont.X = Convert.ToInt32(a0.X + a1.X * u
           + a2.X * u * u + a3.X * u * u * u);
   pont.Y = Convert.ToInt32(a0.Y + a1.Y * u
           + a2.Y * u * u + a3.Y * u * u * u);
   return pont;
}
```

20.32. forráskód.

```
private void erintorajz(Point p0, Point p1,
                        Point t0, Point t1)
{
   Pen p = new Pen(Color.Red);
   try
   {
      g.DrawLine(p, p0.X, p0.Y, t0.X
                    + p0.X, t0.Y + p0.Y);
   }
   catch
   {

   }
   try
   {
      g.DrawLine(p, p1.X, p1.Y, t1.X
                   + p1.X, t1.Y + p1.Y);
   }
   catch
   {

   }
}
```

20.33. forráskód.

```
private void hermrajz(Color poliaszin3)
{
   Point p0, p1;
   p0 = new Point();
   p1 = new Point();
   for (int j = 1; j <= pontdb; j++)
   {
      HermitHatarok(j - 1, j);
      p0 = szak[j - 1];
      p1 = szak[j];
      erintorajz(p0, p1, erinto0, erinto1, 0);
      ep3 = hermiteu3(p0, p1, erinto0, erinto1, 0);
      for (gd = 1; gd <= pontossag; gd++)
      {
         p3 = hermiteu3(p0, p1, erinto0, erinto1, gd / pontossag);
         Pen pen = new Pen(poliaszin3);
         try
         {
            g.DrawLine(pen, ep3.X, ep3.Y, p3.X, p3.Y);
         }
         catch
         {

         }
         ep3 = p3;
      }
   }
}
```

20.34. forráskód.

```
private bool nyom = false;
private const int max = 13;
private Point[] Pontok = new Point[max + 1];
private int n = 0;
private int mozgat = -1;
Graphics g;
```

A fejezet forráskódjai 5.

20.35. forráskód.

```
if(checkBox3.Checked==true)
{
   HermitHatarok(pontdb, 0);
   p0 = szak[pontdb];
   p1 = szak[0];
   erintorajz(p0, p1, erinto0, erinto1);
   ep3 = hermiteu3(p0, p1, erinto0, erinto1, 0);
   try
   {
      for (gd = 1; gd < pontossag; gd++)
      {
         p3 = hermiteu3(p0, p1, erinto0, erinto1, gd / pontossag);
         Pen pen = new Pen(poliaszin3);
         g.DrawLine(pen, ep3.X, ep3.Y, p3.X, p3.Y);
         ep3 = p3;
      }
   }
   catch
   {

   }
}
```

20.36. forráskód.

```
private void polirajz(Color poliaszin)
{
   Point p = new Point();
   Point ep = new Point();
   if (this.checkBox2.Checked == true) hermrajz(Color.Black);
```

```
   if (this.checkBox1.Checked == true)
   {
      Pen pen = new Pen(poliaszin);
      ep = this.HelyiertekBezier(0);
      for (int i = 0; i <= osztas * pontdb; i++)
      {
         double j = (double)i;
         p = this.HelyiertekBezier(j / (osztas * pontdb));
         try
         {
            g.DrawLine(pen, ep.X, ep.Y, p.X, p.Y);
         }
         catch
         {

         }
         ep = p;
      }
   }
}
```

20.37. forráskód.

```
private void pontjelolo(Point[] szak)
{
   Pen pen = new Pen(pontaszin);
   for (int i = 0; i <= pontdb; i++)
      try
      {
         g.DrawRectangle(pen, szak[i].X - 3, szak[i].Y - 3, 6, 6);
      }
      catch
      {

      }
   pen.Color = erintopszin;
   if (this.checkBox4.Checked == false)
   {
      for (int i = 0; i < pontdb; i++)
         try
         {
            g.DrawLine(pen, szak[i].X, szak[i].Y, szak[i + 1].X, szak[i + 1].Y);
         }
         catch
         {

         }
   }
}
```

20.38. forráskód.

```
private void pontjel(Point[] szak, int db)
{
   if (db > 0)
   {
      Pen pen = new Pen(pontpszin);
      g.DrawRectangle(pen, szak[db].X - 3, szak[db].Y - 3, 6, 6);
   }
}
```

20.39. forráskód.

```
private void poliujra()
{
   ClearDevice();
   pontjelolo(szak);
   aktp = -1;
   pontmozg = -1;
   polirajz(poliaszin);
   this.numericUpDown1.Value =
          Convert.ToDecimal(erszor * 10);
}
```

20.40. forráskód.

```
private void segedvonal(Point[] szak, int sz, Color szin)
{
   if (sz != 0 && sz != pontdb)
   {
      Pen pen = new Pen(pontaszin);
      try
      {
         g.DrawRectangle(pen, szak[sz].X
                - 3, szak[sz].Y - 3, 6, 6);
      }
      catch
      {

      }
      if (this.checkBox4.Checked == false)
      {
         pen.Color = szin;
         try
         {
```

```
            g.DrawLine(pen, szak[sz].X, szak[sz].Y,
                        szak[sz - 1].X, szak[sz - 1].Y);
         }
         catch
         {

         }
         try
         {
            g.DrawLine(pen, szak[sz].X, szak[sz].Y,
                        szak[sz + 1].X, szak[sz + 1].Y);
         }
         catch
         {

         }
      }
   }
   if (sz == 0)
   {
      Pen pen = new Pen(pontaszin);
      try
      {
         g.DrawRectangle(pen, szak[sz].X - 3,
                        szak[sz].Y - 3, 6, 6);
      }
      catch
      { }
      pen.Color = szin;
      if (this.checkBox4.Checked == false)
      {
         try
         {
            g.DrawLine(pen, szak[sz].X, szak[sz].Y,
                        szak[sz + 1].X, szak[sz + 1].Y);
         }
         catch
         { }
      }
   }
   if (sz == pontdb)
   {
      Pen pen = new Pen(pontaszin);
      try
      {
         g.DrawRectangle(pen, szak[sz].X - 3,
                        szak[sz].Y - 3, 6, 6);
      }
      catch
      { }
      if (this.checkBox4.Checked == false)
      {
         pen.Color = szin;
         try
         {
            g.DrawLine(pen, szak[sz].X, szak[sz].Y,
                        szak[sz - 1].X, szak[sz - 1].Y);
         }
         catch
         { }
      }
   }
}
```

20.41. forráskód.

```
private void initrajz()
{
   pontmozg = -1;
   aktp = -1;
   szak[0].X = 50; szak[0].Y = 200;
   szak[1].X = 100; szak[1].Y = 150;
   szak[2].X = 150; szak[2].Y = 120;
   szak[3].X = 200; szak[3].Y = 150;
   szak[4].X = 250; szak[4].Y = 200;
   pontdb = 4;
   this.numericUpdown1.Value =
          Convert.ToDecimal(erszor * 10);
}
```

20.42. forráskód.

```
private void Form1_Paint(object sender,
                  PaintEventArgs e)
{
   g = e.Graphics;
   pontjelolo(szak);
   polirajz(poliaszin);
}
```

A fejezet forráskódjai 6.

20.43. forráskód.

```
private void Form1_MouseMove(object sender,
                        MouseEventArgs e)
{
   if (pontmozg > -1)
```

```
   {
      segedvonal(szak, aktp, erintoaszin);
      polirajz(poliaszin);
      szak[aktp].X = e.X;
      szak[aktp].Y = e.Y;
      Refresh();
      polirajz(pontaszin);
      segedvonal(szak, aktp, erintoaszin);
   }
}
```

20.44. forráskód.

```
private void Form1_MouseDown(object sender, MouseEventArgs e)
{
   if (e.Button == MouseButtons.Left)
   {
      j = 0;
      if (aktp > -1)
      {
         if ((e.X >= szak[aktp].X - 3 && e.X
               <= szak[aktp].X + 3) &&
                        (e.Y >= szak[aktp].Y - 3 &&
            e.Y <= szak[aktp].Y + 3))
            pontmozg = aktp;
         else j = -1;
         if (j == -1)
         {
            segedvonal(szak, aktp, erintoaszin);
            segedvonal(szak, aktp, erintopszin);
            aktp = -1;
         }
      }
      if (aktp == -1)
      {
         j = -1;
         for (gm = 0; gm <= pontdb; gm++)
            if (((e.X >= szak[gm].X - 3) &&
                        (e.X <= szak[gm].X + 3)) &&
               ((e.Y >= szak[gm].Y - 3) &&
                        (e.Y <= szak[gm].Y + 3)))
               j = gm;
         if (j > -1)
         {
            aktp = j;
            pontmozg = j;
            segedvonal(szak, j, erintopszin);
            segedvonal(szak, j, erintoaszin);
            if (this.checkBox5.Checked == true)
            {
               for (int i = j; i <= pontdb - 1; i++)
               {
                  szak[i].X = szak[i + 1].X;
                  szak[i].Y = szak[i + 1].Y;
               }
               pontdb--;
            }
         }
      }
   }
   if (e.Button == MouseButtons.Right && aktp == -1)
   {
      if (pontdb < maxp)
      {
         polirajz(poliaszin);
         pontdb++;
         szak[pontdb].X = e.X; szak[pontdb].Y = e.Y;
         polirajz(poliaszin);
         segedvonal(szak, pontdb, erintopszin);
      }
   }
   Refresh();
}
```

20.45. forráskód.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
   g = e.Graphics;
   if (n > 0)
   {
      Gorbe();
      PontKi(n);
   }
}
```

20.46. forráskód.

```
private void Form1_MouseUp(object sender,
                  MouseEventArgs e)
{
   pontmozg = -1;
   poliujra();
}
```

20.47. forráskód.

```
private void button1_Click
                (object sender, EventArgs e)
{
   Application.Exit();
}

private void Form1_Load(object sender, EventArgs e)
{
   initrajz();
}

private void checkBox2_CheckedChanged
             (object sender, EventArgs e)
{
   Refresh();
}

private void numericUpDown1_ValueChanged
            (object sender, EventArgs e)
{
   erszor = Convert.ToDouble
            (this.numericUpDown1.Value) / 10;
   Refresh();
}
```