C# a gyakorlatban

Dr. Illés Zoltán, H. Bakonyi Viktória

A kiadvány „A felsőoktatásba bekerülést elősegítő készségfejlesztő és kommunikációs programok megvalósítása, valamint az MTMI szakok népszerűsítése a felsőoktatásban” (EFOP-3.4.4-16-2017-006) című pályázat keretében készült 2018-ban.

Eötvös Loránd Tudományegyetem
Informatikai Kar

C#: egy nyelv – ezernyi lehetőség

C# a gyakorlatban

Szerző

Dr. Illés Zoltán, H. Bakonyi Viktória

Felelős kiadó

ELTE Informatikai Kar
1117 Budapest, Pázmány Péter sétány 1/C.

ISBN szám

ISBN 978-963-489-038-6

A kiadvány „A felsőoktatásba bekerülést elősegítő készségfejlesztő és kommunikációs programok megvalósítása, valamint az MTMI szakok népszerűsítése a felsőoktatásban” (EFOP-3.4.4-16-2017-006) című pályázat keretében készült 2018-ban

Tartalomjegyzék

Bevezetés

Ma, amikor elterjedőben van az a nézet, hogy az alapműveltség része a programozás, rá kell ébrednünk, hogy akármilyen mélységben is sajátítunk el egy programozási nyelvet, az nem elegendő ahhoz, hogy elmondhassuk, tudunk programozni. Gondoljuk csak végig, hogy mire lehet még szükség az alkotómunkához!

Az első programok megszületése idején maguk a számítógépek is jóval kezdetlegesebbek voltak, így erőforrásaik nem is voltak elegendők komolyabb méretű feladatok végrehajtására. A számítások végrehajtásának gyorsasága volt, ami vonzóvá tette használatukat a szakemberek számára, így nem kellett többek között sem nagyon nagyméretű kódokkal, sem pedig igényes felhasználói felülettel foglalkozni. Mára a helyzet alaposan megváltozott, jóval összetettebb feladatok elvégzése is lehetségessé vált, ráadásul hozzászoktunk a minket körülvevő multimédiás környezethez. Gondoljunk csak például a szinte élethű grafikával jellemezhető számítógépes játékokra, a valós idejű forgalominformációkat megjelenítő appokra, a már néhol megjelenő emberi tulajdonságokkal is rendelkező robotokra vagy az okos városokra. Az ilyen és az ehhez hasonló bonyolultsággal rendelkező feladatok megvalósításához egy programozási nyelv és megfelelő algoritmikus gondolkodáson túl nyilvánvalóan egyéb ismeretekkel is rendelkeznünk kell!

Elmondható, hogy a nagyobb méretű kódok megjelenésével újabb és újabb programszervezési módszereket vezettek be, hogy átláthatóbbá tegyék a munkát. Elsőként megjelentek az eljárások, függvények, a következő lépés a kód modulokba szervezése lehetett és a különböző könyvtári szolgáltatások alkalmazása. Még magasabb szintű szervezés a többrétegű architektúra létrehozása, amelyek közül ma a legelterjedtebb a háromrétegű alkalmazások világa, ahol a rétegek egymástól szinte teljesen függetlenül készülhetnek (megjelenítés, üzleti logika és adatréteg). Más, de hasonlóan rétegekbe szétosztott program architektúra az *MVC* (modell, view, controller) modell. Ezek a rétegek akár különböző számítógépeken is futhatnak: a megjelenítés egy kliens gépen történik, az üzleti logika egy szerveren fut és az adatréteget pedig egy adatbázis szerver biztosítja. Ma gyakran *web-szolgáltatásokat* alakítanak ki és azok meghívásával juthatunk hozzá a távoli adatokhoz. Ezek a web-szolgáltatások gyakran a *felhő*ben futnak.

Természetes az is, hogy egy feladatot ma nem egyetlen ember, hanem több programozó együttesen (esetleg egyéb szakemberekkel kiegészítve például grafikussal) készítik. A programtervező informatikusok feladata a leendő alkalmazás megtervezése, amit ma már programok segítségével végezhetnek. (Ezek a lehetőségek a tervből egy alapkód legenerálását is el tudják végezni igény esetén, de nagy segítséget nyújtanak a kód analizálásában is. Pl. Vs Architecture) A szoftverfejlesztő cégek a közös munka hatékony megszervezéséhez kialakítottak néhány módszertant, mint például a vízesés jellegű illetve a különböző agilis módszertanokat. A választott módszertantól függetlenül elmondható, hogy gyakorlatilag egy nagyobb, többemberes feladatnál lehetetlen volna a fejlesztők által elkészített programkódok verzióinak kezelése, összefésülése kézi eszközökkel. Emiatt aztán létszükségletté váltak a különböző *verziókezelő-rendszerek* és mára ezek az eszközök a hobbi programozók kezében is fontos lehetőséggé váltak.

Mit sem érnek azonban a különböző verziók, ha azokat nem teszteljük le alaposan. Mindannyian ismerjük a tesztadatok kiválasztásának fekete és fehérdoboz módszereit. Ma a modern fejlesztőeszközök beépített *programtesztelési* lehetőségekkel pl. unit-tesztelési modulokkal rendelkeznek. Elterjedt a tesztvezérelt programozás fogalma is, amikor még az alkalmazás elkészülte előtt létrehozzák a teszteket és a majdani programnak ezeken az előre elkészített teszteken kell jól teljesítenie.

A szoftverkészítők életét tovább bonyolítja, hogy az elkészült alkalmazásokat a felhasználók számtalan különböző platformon kívánják használni. Természetes igénnyé vált, hogy bármelyik eszközünket használjuk is, ugyanazokat a lehetőségeket érhessük el, a különböző rendelkezésre álló erőforrásoktól függetlenül, akár kis vagy óriási kijelzővel rendelkezünk, akár billentyűzetről, egérrel vagy érintőképernyővel van dolgunk. Gondoljunk csak arra, hogy okos TV-vel is internetezhetünk, mobil telefonon is nézhetünk filmeket! Legyen bármekkora is egy cég, túlságosan nagy erőforrást köt le, ha minden alkalmazást bárhonnan elérhetővé akarnának tenni (nem beszélve arról, ha mi magunk akarunk egy több platformon futó programot készíteni). Ennek az igénynek a kielégítésére születnek, születtek a különböző *multiplatform*os megoldások. Ilyen irány a Windows Universal App lehetőség, amellyel elvileg minden Windowst futtató rendszeren ugyanaz az alkalmázás használható vagy a Xamarin, amely lehetővé teszi különböző operációs rendszerrel rendelkező mobil telefonokra is a megosztott kód használatát.

A fentiekben igyekeztünk felvázolni napjaink programfejlesztéssel kapcsolatos főbb lehetőségeit, gyakorlatát. Visszatérve a kiinduló gondolatunkhoz, valóban nem elegendő egy programozási nyelv, annak, szintaktikájának ismerete ahhoz, hogy összetett, jó programokat tudjunk készíteni a gyakorlatban vagy bekapcsolódjunk egy fejlesztő csapat munkájába, de a C# nyelv lehetőségeivel felvértezve és a Visual Stúdió professzionális segítségével mindezeket a praktikákat is hamarosan a magunkévá tehetjük.

Rajta, tegyük meg együtt az első lépéseket a gyakorlati programozás irányába!

Megjegyzés

A megoldásokat Visual Studio 2017 Enterprise (verzió 15.5.6.) segítségével készítettük, 4.6.1. .Net Framework-öt használva.

I. Modern programkészítési eszközök

Mielőtt belekezdenénk a feladatok megoldásába, tekintsük át azt az eszközkészletet, amely segít majd bennünket a kódolási munkában.

Verziókezelés

Ahogy azt korábban említettük, mára már elterjedté váltak az úgynevezett verziókezelő rendszerek. Hasznosságuk abban mutatkozik meg, hogy egyfelől lehetővé teszik, hogy megőrizzünk több különböző verziót, amelyekhez visszaléphetünk, másfelől a több programozó által készített párhuzamos megoldások összeépítésére is alkalmasak. (Két fajtájuk van az elosztott és a központosított – minderről kicsit bövebben a https://bit.ly/2DfOopo linken olvashat.) De sokan itt tesznek közzé publikusan is kódokat, kódrészleteket, amelyeket szabadon felhasználhatunk.

A könyv feladatai a GitLab (https://about.gitlab.com/) rendszeren keresztül lesznek elérhetőek, de mivel ingyenesen használható eszközről van szó, érdemes mindenkinek a megoldások letöltése mellett megfontolni a saját célú felhasználását is. Ennek alapjain fogjuk most végigvezetni a tisztelt olvasót!

Kétféleképpen kezelhetjük az állományaink verzióit, egyfelől parancssorból parancsok kiadásával (segítség ehhez a https://desoft.hu/downloads/git/git_v1.0.pdf linken olvasható), illetve a Visual Studió-n keresztül is. Ez utóbbi használatát mutatjuk be röviden, mert a kódolási munkát is ebben a fejlesztőkörnyezetben végezzük.

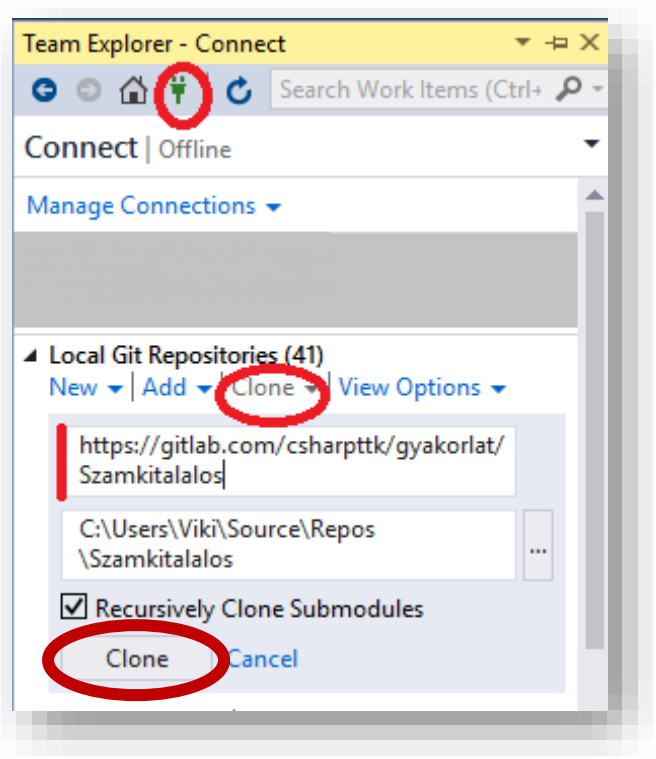
A feladatok letöltése.

Az anyagban a feladatok után letöltési címet találhatunk ilyen formában. (Ez a következő fejezet legelső feladata.)

Ez a letöltési cím!

Letölthető	
Számkitalálós példa	https://gitlab.com/csharpttk/gyakorlat/Szamkitalalos

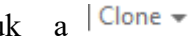
Indítsuk el a Visual Studio-t, majd nyissuk meg a Team Explorer fület.



A  ikont kiválasztva eljutunk a Git Repository-khoz.

Megjegyzés:

Ezen a fülön a Microsoft Team Foundation Serveréhez is kapcsolódni lehetne. Ez a lehetőség maximum 5 fő együttes munkáját teszi lehetővé, ezért iskolai munkához kevésbé alkalmas.

Ezután válasszuk a 

menüpontot és másoljuk be a feladatok végén megadott letöltési címet.

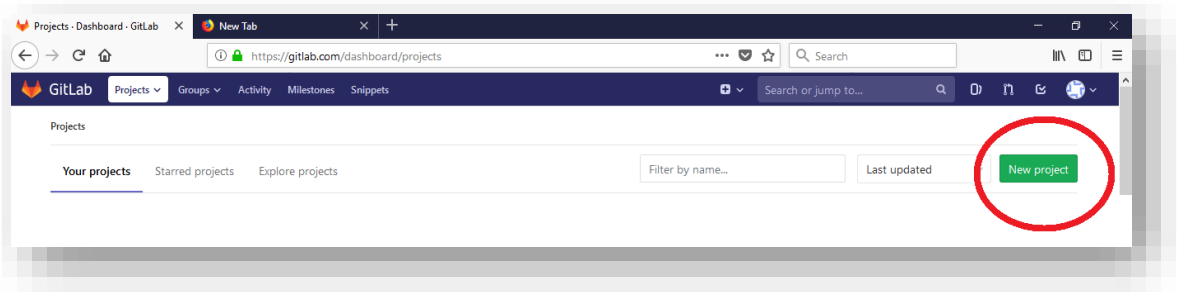
Alapértelmezettként megadott lokális mappát kívánság szerint megváltoztathatjuk, ha ezt nem tesszük, akkor a letöltött megoldások a c:\Users\saját_azonosító/Source/Repos mappába kerülnek.

A 

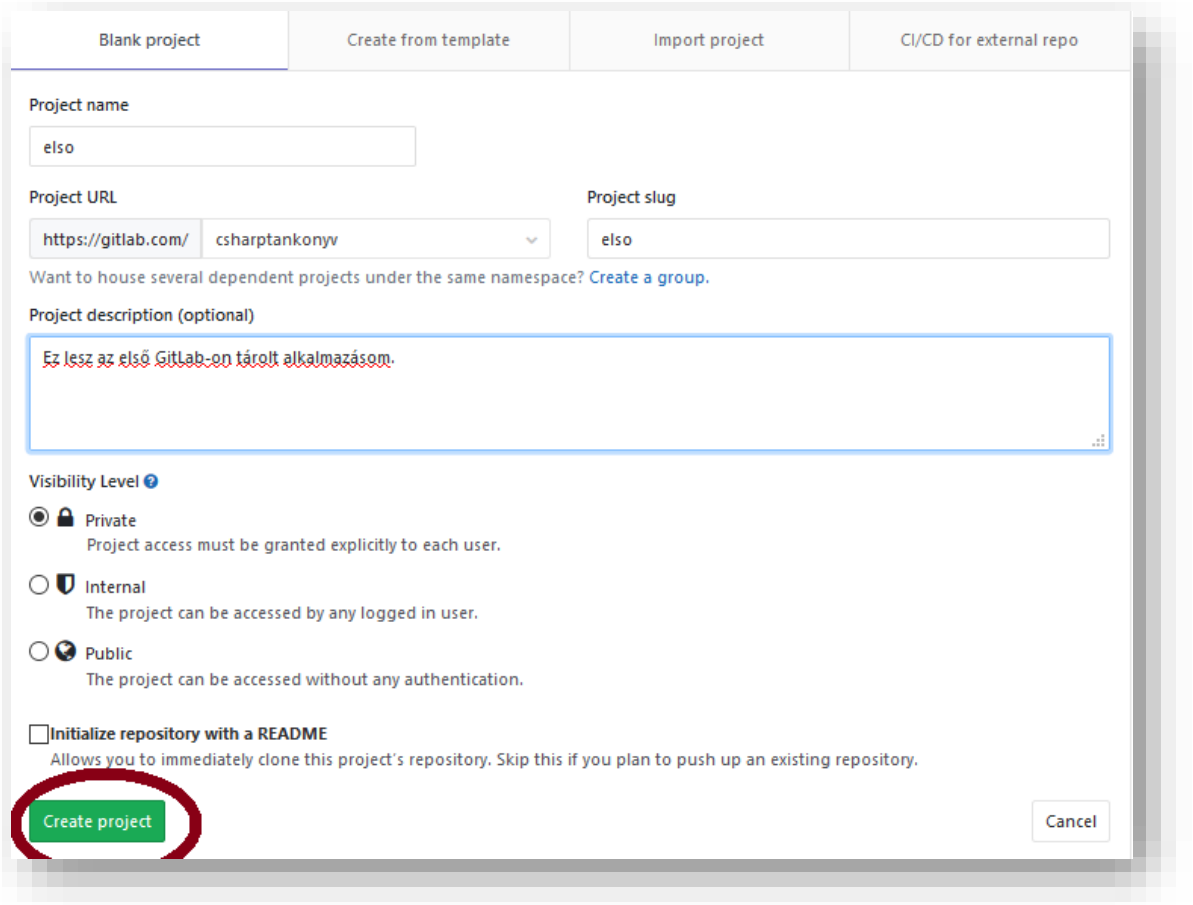
gomb lenyomására megtörténik a letöltés és az alkalmazás használatra kész.

GitLab használata.

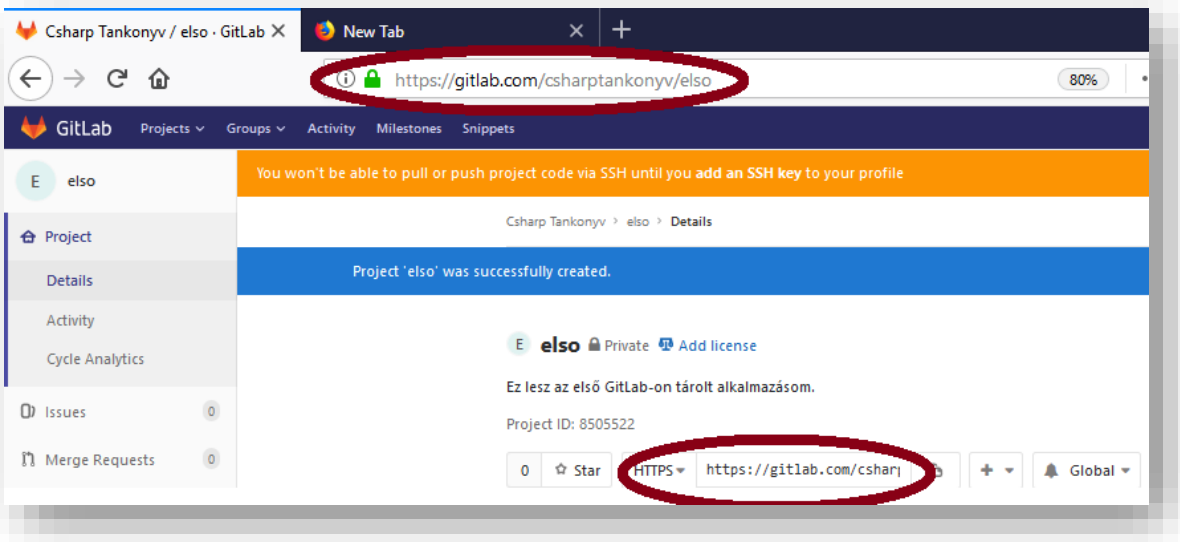
Verziókezelés vagy akár kódmegosztás céljából is érdemes lehet az iskolai órákon is használni a GitLab-ot. Regisztrálnunk kell a GitLab-ra (https://gitlab.com/users/sign_in), ahova szükség lesz egy e-mail címre.

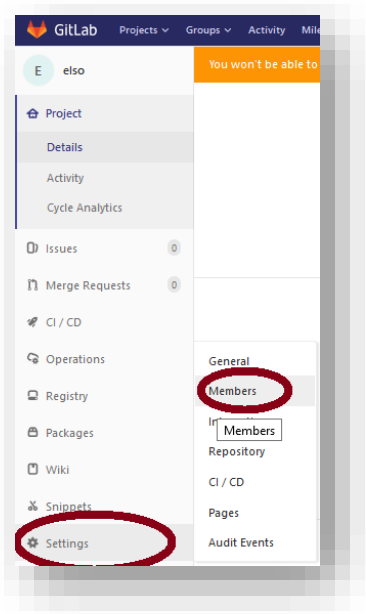


Készítsünk egy projektet mondjuk elso néven – ehhez kattintsunk a New project gombra, majd adjuk meg a nevet és válasszuk ki a hozzáférési jogosultságot. Jelen esetben Private hozzáférést választottunk – ilyenkor csak a hozzárendelt felhasználók érhetik el az alkalmazást.



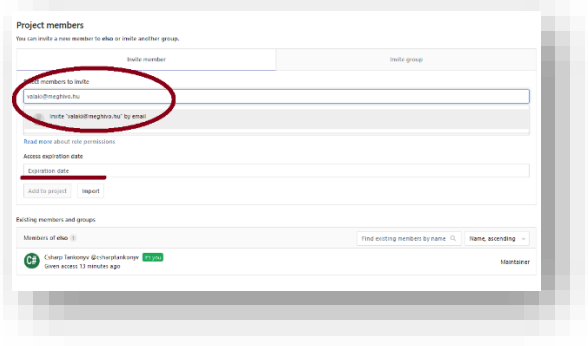
Miután lenyomtuk a Create project gombot, az alkalmazás létrejön (egyelőre üresen) és elérhetőségét két helyen is láthatjuk a következő ablakban. Jelen esetben ez https://gitlab.com/csharptankonyv/elso link lesz.





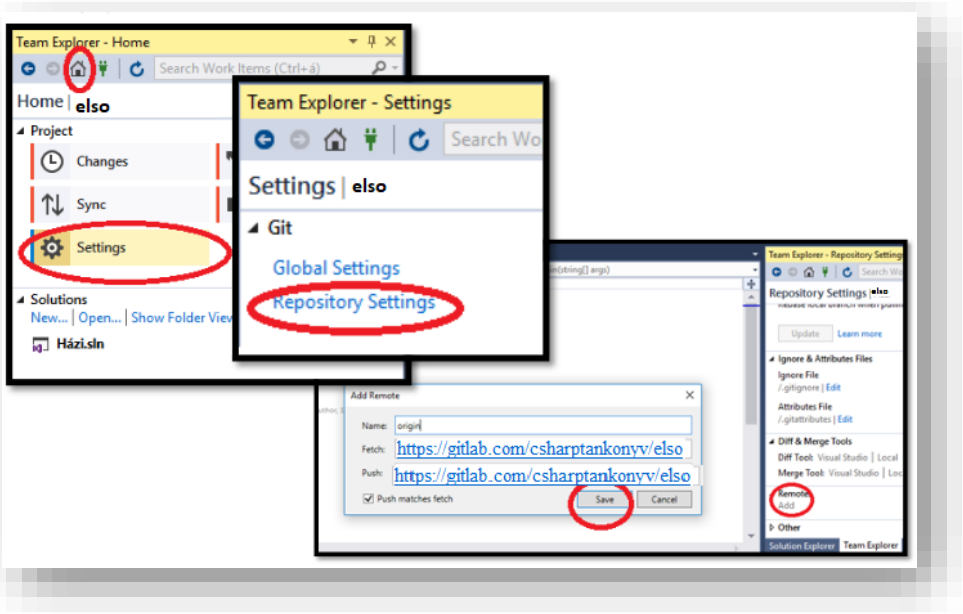
A létrejött Private alkalmazáshoz megadhatjuk az azt kezelők körét. Válasszuk a Settings, majd a Members menüpontot!

Adjuk meg vagy az e-mail címét, vagy a GitLab azonosítóját az új tagnak. (Neki is kell majd GitLab azonosító.), adjuk meg a lejárati időt és a szerepkört. Reporter szerepkörben csak letöltési joga lesz. Bővebben a szerepkörökről a https://gitlab.com/help/user/permissions címen olvashat.



Hozzunk létre mondjuk egy konzolalkalmazást az elindított Visual Studio-ban. Kérjük, hogy hozzon létre hozzá Git repository-t is, amit majd a Team Server ablakon keresztül kapcsolunk hozzá a GitLab-on létrehozott projekthez.

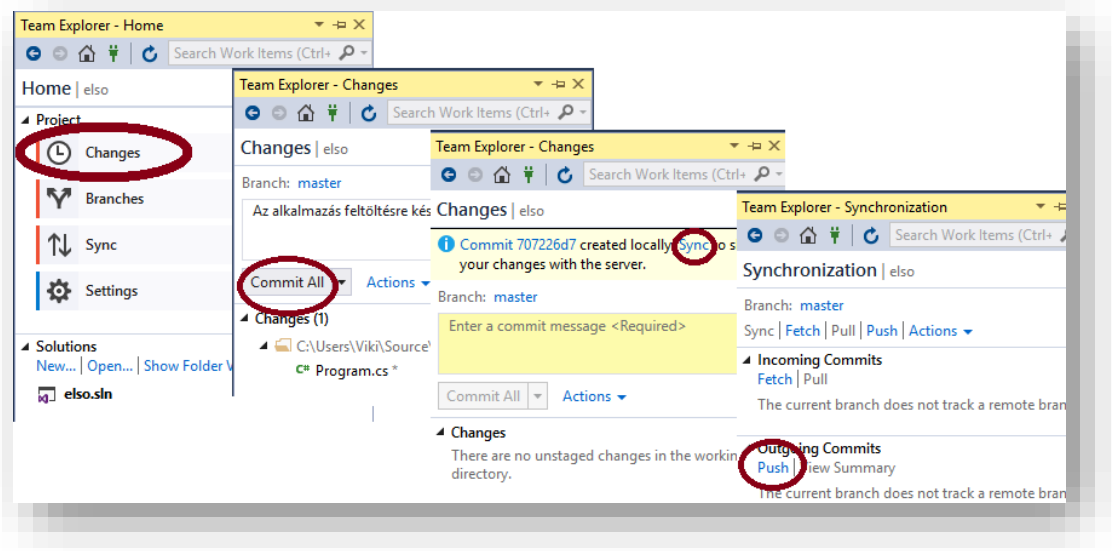
A lokálisan létrejött alkalmazás és a GitLab összekapcsolását a következő módon tegyük meg.



Nyissuk meg a Team Explorer ablakot és kattintsunk a  ikonra, majd válasszuk a Settings lehetőséget, azon belül a Repository Settings-et. Folytassuk a Remote alatti Add gomb lenyomásával, majd a felbukkanó ablakban adjuk meg az origin szót illetve másoljuk be a GitLab-on megadott elérési utat, majd mentsük el. Az összekapcsolást elkészítettük.

Készítsük el az alkalmazásunkat, majd töltsük fel a GitLab-ra, amit két lépcsőben végzünk el.

Először a munkaterületről a változásokat átadjuk a „stage”-nek, ahonnan majd a tényleges feltöltés megtörténik.



A lépések a következőek: Team Explorer ablakban válasszuk a Changes menüpontot. Adjuk meg az aktuális verzió leírását (mit változtattunk az előzőekhez képest.), *commit*-áljuk, azaz küldjük át a szinkronizálásra várakozó területre. Végezetül indítsuk el a szinkronizálást és a *push* lehetőséggel töltsük fel a változásokat.

Megjegyzés.

Ha újabb verziókat akarunk feltölteni, a lépéssorozat ugyanez lesz. Ha a szerveren levő újabb verziókat akarjuk letölteni, akkor a szinkronizáció utolsó lépésében a pull lehetőséget használjuk!

Megjegyzés:

Természetesen ennél jóval több lehetőség áll rendelkezésre a verziókezelőben, például egy projekt különböző ágainak (*branch*) kezelése, több ág összeépítése (*merge*) vagy visszalépés egy előző verzióhoz (*revert*).

Kódgenerálás

Az algoritmusból kód generálás több fejlesztőeszköz része (pl. a sokak által ismert ingyenes CodeBlocks-é is). Lényege, hogy a terv, a struktogram, az UML elkészülése lehetővé teszi a kód vázának automatikus legenerálását felgyorsítva a kódolást.

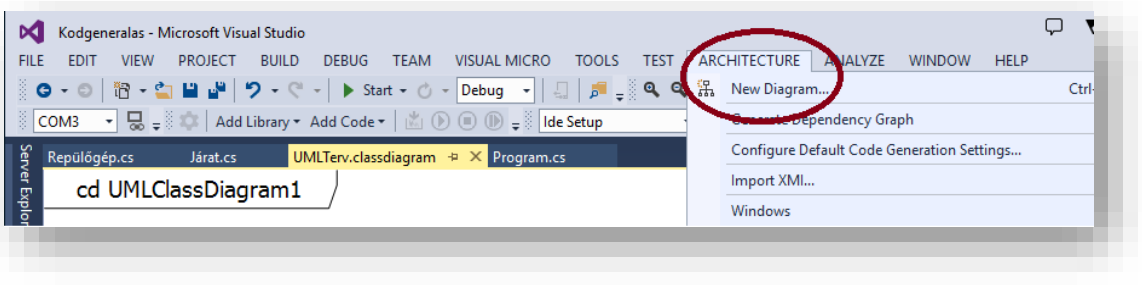
Feladat:

Hozzunk létre két osztályt egy repülő és egy járat osztályt. Tervezzük meg az attribútumokat és eljárásokat, majd kódoljuk is le ezeket.

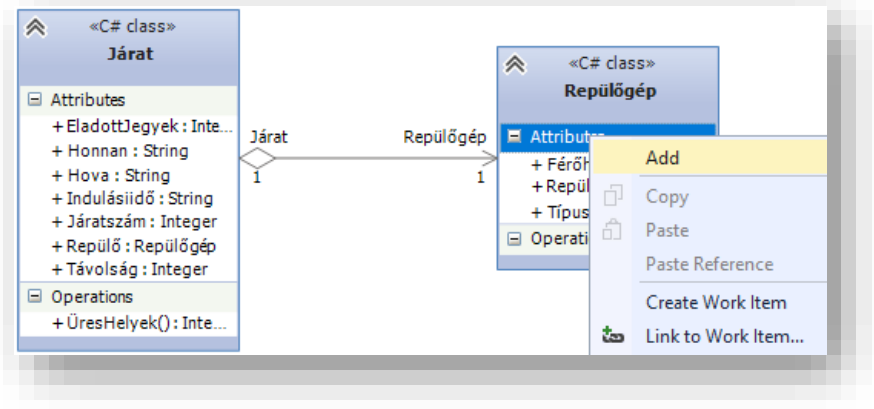
Megjegyzés

Ez a szolgáltatás az oktatásban aláhúzhatja a tervezés jelentőségét és meggyőzheti a diákokat arról, hogy érdemes a tervezéssel kezdeni a munkát. Sajnos a Visual Studio 2017-ből kikerült ez a szolgáltatás, de egy villanás erejéig megmutatjuk a korábbi verzióban hogy is történt ez, hogy jobban el lehessen képzelni a folyamatot.

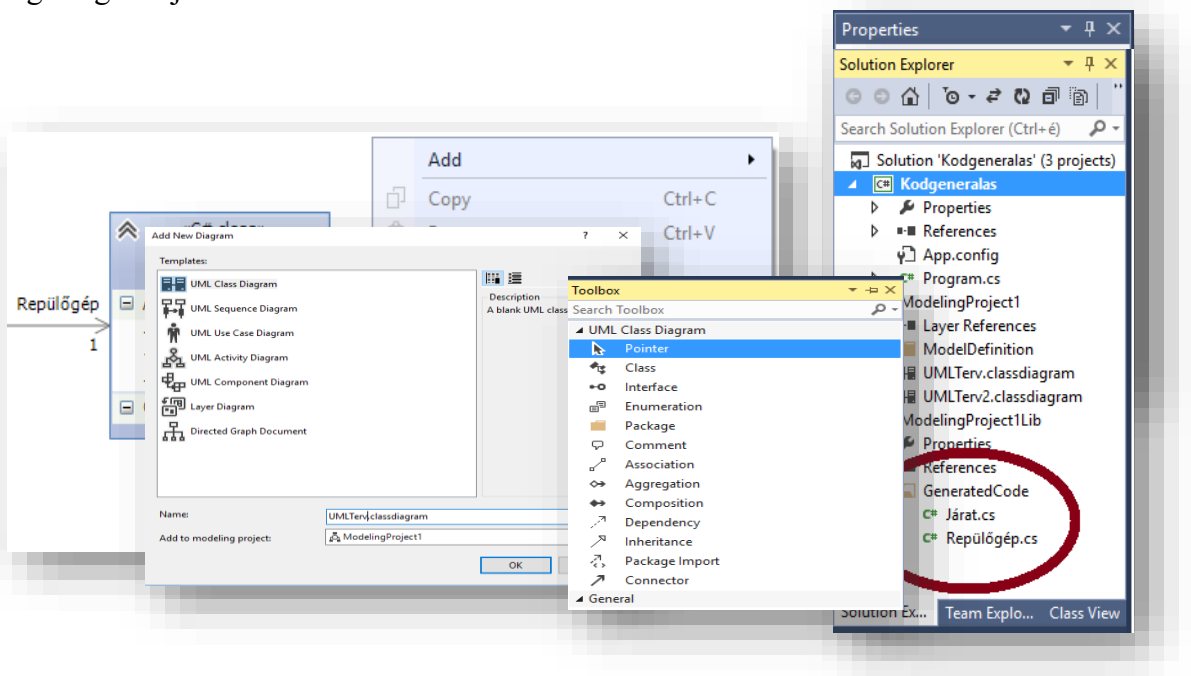
Hozzunk létre akár egy konzol alkalmazást, majd az Architecture menüben válasszuk a New Diagram menüpontot. Ezután ki kell választanunk, hogy milyen UML ábrát szeretnénk megtervezni – ez most UML osztályterv.



Ezután nincs más dolgunk, mint a Tool ablakból a megfelelő elemekből felépíteni az osztályokat úgy, hogy ráhúzzuk őket az ablakra!



Az osztályokhoz attribútumokat és metódusokat rendelhetünk a property ablakban beállítva a tulajdonságaikat. A példában egy repülőgép és egy járat osztály UML diagramját készítettük el. Az ábrán látható, hogy jobb egérgombra újabb és újabb elemekkel bővíthetjük az osztályunkat az *Add* menüponttal és a property ablak egy részletét is megvizsgálhatjuk.

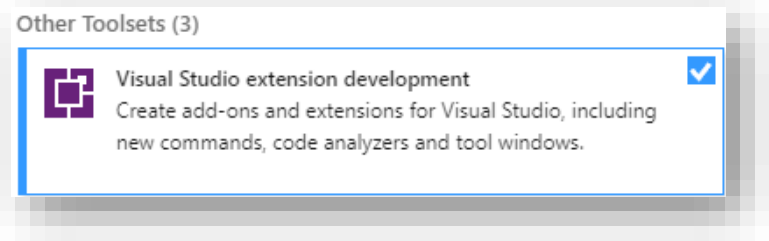


A következő lépésben legeneráltathatjuk a kódot a pop-up menüből. A két osztálynak megfelelő kódfájlok megjelennek a Solution Explorerben már csak tovább kell lépni az alapokból!

Kód áttekintése

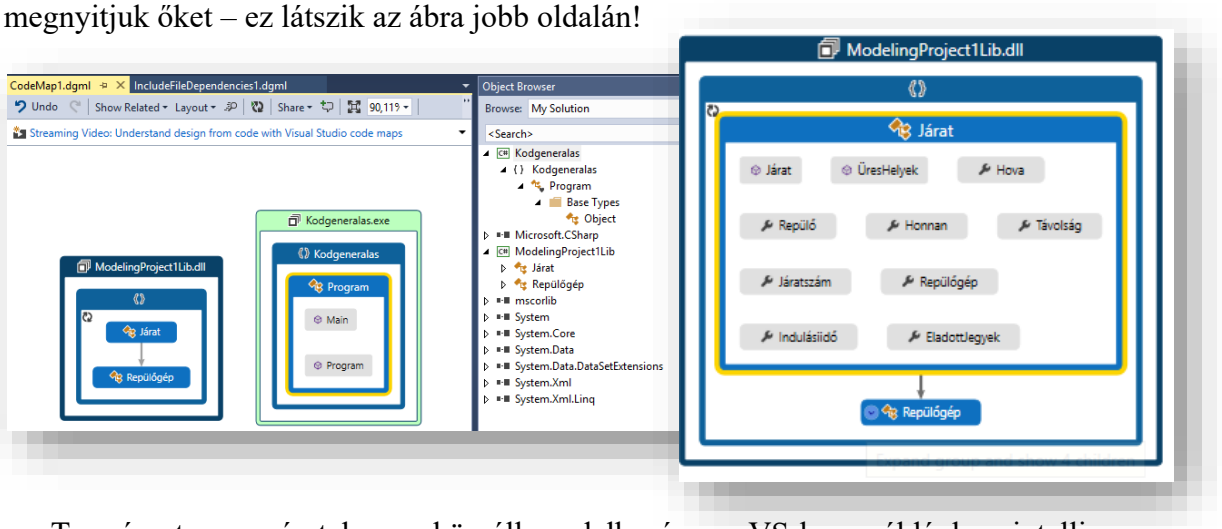
Egy komplex alkalmazás áttekintéséhez viszont szükség van arra, hogy az elkészült kód részei közötti összefüggések, az osztályok felépítése és a köztük levő kapcsolatokat a projektvezető könnyebben átlássa. A *Code Map* szolgáltatás része a Visual Studio 2017-nek is.

Megjegyzés:



A Code Maps lehetőséghez a Visual Studio Installer-ben telepíteni kell a és ki kell választani az jobb oldalon az Architecture and analysis tools-t.

Az előbb bemutatott osztály tervezéses feladatot nyitottunk meg a Studio-ban, majd az *Architecture* menüben a *New Code Map* menüpontot választottuk. Ezután erre húzhatjuk rá például a projektben lévő osztályokat. Tartalmukat részletesebben is megvizsgálhatjuk, ha megnyitjuk őket – ez látszik az ábra jobb oldalán!



Természetesen számtalan eszköz áll rendelkezésre a VS-ben, például az intellisence, vagy a code lence, de ezek a lehetőségek nem a tervezés, inkább a közvetlen kódolás segítői.

Tesztelés

Mit sem ér az a program, amelyről nem tudjuk, hogy helyesen működik-e - a bemenő adatokra az elvárt kimeneteket produkálja-e. Korábban már említettük, hogy ma egyre inkább az automatikus tesztelés a mérvadó. Ennek előnye, hogy a sajnos a program legapróbb változtatása után újra le kellene futtatni a teszteket, hogy meggyőződjünk arról, nem rontottuk-e el a kódolást valahol.

Nyilvánvalóan ennek lebonyolítása újra meg újra automatizálás nélkül rengeteg időbe kerülne. Emiatt ma automatizált teszteléseket alkalmaznak, vagyis programot készítenek a teszteléshez is! A Visual Studio tartalmaz úgynevezett unit tesztelési lehetőséget, amely arra szolgál, hogy egy-egy kisebb rész automatikus tesztelését el tudjuk végezni. Nézzük meg egy egyszerű példán keresztül, hogy hogyan is kell elképzelni ezt a mechanizmust!

Feladat:

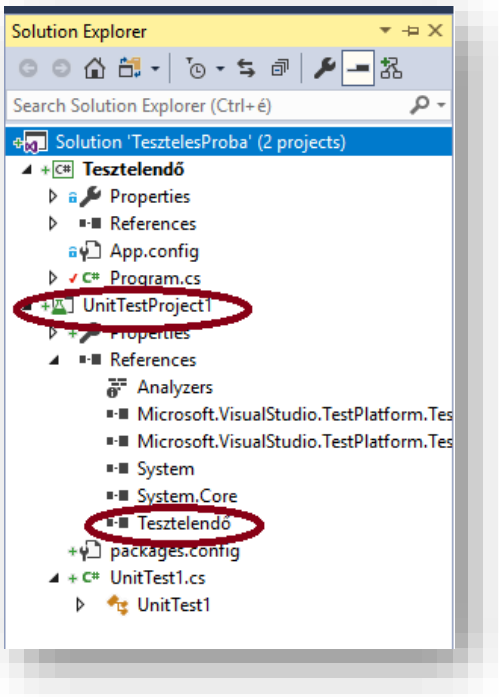
Állapítsuk meg egy szóról, hogy eszperente szó-e, azaz csak e-t tartalmaz a magánhangzók közül. Készítsünk ehhez egy egyszerű C# konzol alkalmazást, amely a *csupae* függvényt valósítja meg!

Példa

```
public static bool csupae(string szó)
        {
            szó = szó.ToLower();
            string mgh = "aáéiíoóöőuúüű"; //ez e betű nincs benne...
            int i =0  ;
            while (i < mgh.Length && !szó.Contains(mgh[i]))
                i++;
            return i >= mgh.Length;
        }
```

A megoldást tartalmazó projekthez adjunk hozzá egy Unit test alkalmazást és ebben referenciaként adjuk hozzá a tesztelendő alkalmazásunkat.

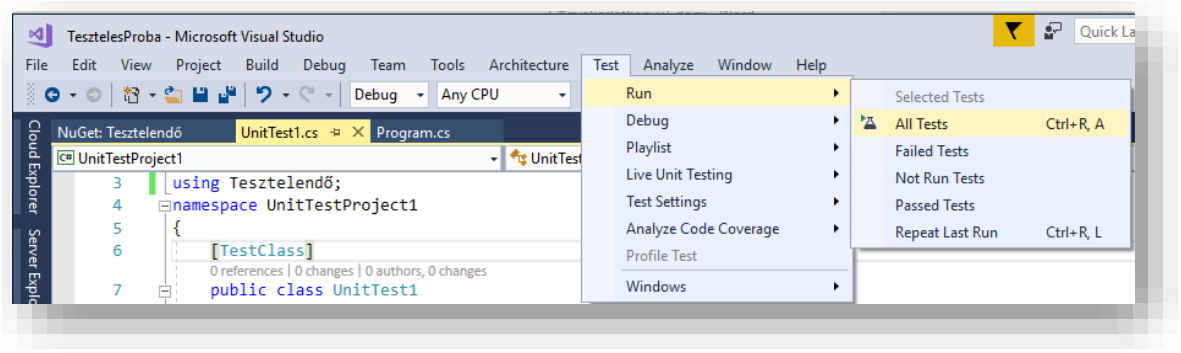
A unit teszt projektben készítsük el az alábbiak szerint az egyes tesztesetekhez tartozó kódot.



Példa

```
[TestClass]
public class UnitTest1
{
    [TestMethod]
    public void TestMethod1()
    {
       string szó = "alma";
       bool eredmény = Program.Csupae(szó);
       Assert.AreEqual(false, eredmény);
    }
    …
```

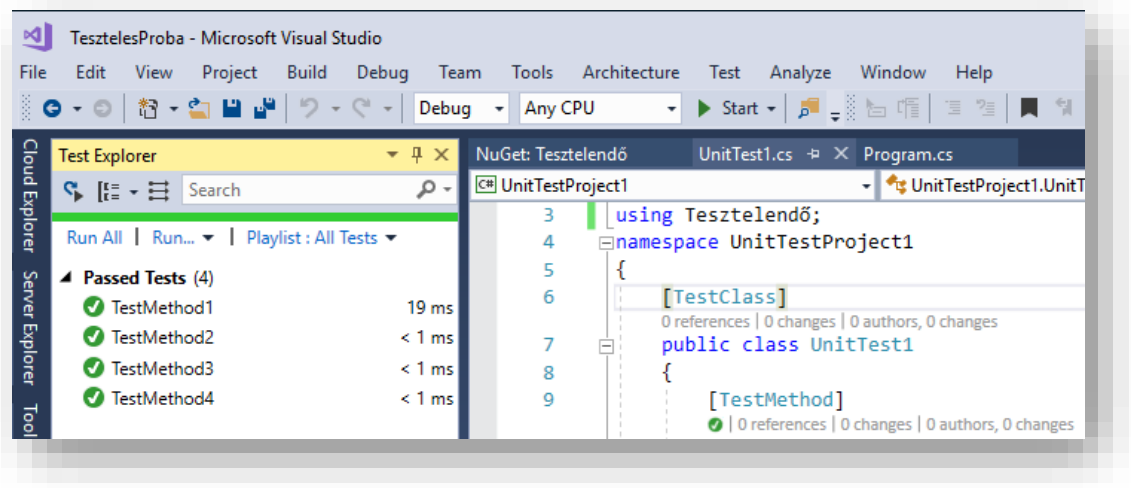
Az *Assert.AreEqual*(false,eredmény) meghívásával hasonlítjuk össze az elvárt és a teszt alapján kapott eredményt. A teszteket a *Test* menüpont *Run* alpontján keresztül érhetjük el. Az *All Tests* hatására a Visual Stúdió lefuttatja az összes tesztet és kiértékeli azt.



A tesztek kiértékelése a Test Explorer ablakban jelenik meg, ahogy az az ábrán látszódik.

Megjegyzés

Természetesen az itt bemutatott lehetőségnél jóval nagyobb eszközkészletet kapunk kézhez a Visual Studio használatával.



Letölthető	
Tesztelési feladat	https://gitlab.com/csharpttk/gyakorlat/teszteles

II. Alapozó ismeretek

Megjegyzés:

A feladatmegoldások során az első kötet (A C# nyelv lehetőségei) 2. és 3. fejezetének elméleti ismereteire fogunk támaszkodni.

a) Alap nyelvi elemek

1. Feladat:

Kezdjünk egy egyszerű számkitalálós feladattal! Először konstansként tároljuk kedvenc számunkat (ez titok marad majd a felhasználók előtt ), majd addig olvassunk be tippeket, amíg végül el nem találják. Készítsünk egy C# nyelvű konzolos alkalmazást!

A megoldásban előforduló új nyelvi elemek: változó deklaráció, ciklus szervezés, elágazás, beolvasás, kiírás, konverzió.

Megoldási ötlet:

Ciklussal addig olvasunk be tippeket, amíg a felhasználó el nem találja a kívánt számot. Segítségül eláruljuk, hogy kisebb vagy nagyobb értéket adott-e meg.

Példa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApp3
{
    class Program
    {
        static void Main(string[] args)
        {
            const int szám = 17;
                 // Random r=new Random(); int szám = r.Next(1, 100);
            int tipp = -1;
            while ( tipp != szám )
            {
                tipp = Convert.ToInt16(Console.ReadLine());
                if ( tipp < szám ) { Console.WriteLine("Túl kicsi"); }
                if ( tipp > szám )  { Console.WriteLine("Túl nagy"); }
            }
            Console.WriteLine("Gratulálok, eltalálta");
        }
    }
}
```

Gondolkodjon el!

Miért használjuk a static kulcsszót?

Megjegyzés:

Ne felejtsük el, hogy a beolvasás Console.ReadLine( ) függvénye string típust ad vissza, így használnunk kell a Convert.ToInt16 függvényt, hogy egészként tárolhassuk az értéket.! (Ha nem egész számot olvasunk be, programunk egyelőre hibás, de később orvosoljuk ezt a gondot!)

Megjegyzés:

A konstans szám helyett véletlenszámot is generálhatunk: **Random r=new Random; szám=r.Next(1,100)** beírásával!

Letölthető	
Számkitalálós példa	https://gitlab.com/csharpttk/gyakorlat/Szamkitalalos

2. Feladat:

Játsszunk egy leegyszerüsített akasztófa játékot (nem fogunk most rajzolni)! Ki kell találni egy szót úgy, hogy egyszerre csak egyetlen betűt adhatunk meg tippként. Ha van benne ilyen, akkor megkapjuk a szóban a betű helyét, ha nincs akkor hibapontot gyűjtöttünk és 9 hiba után veszítünk. Készítsünk egy C# nyelvű konzolos alkalmazást!

A megoldásban előforduló új nyelvi elemek: string és char típus, szövegkezelő függvények, függvény készítés és paraméterek.

Megoldási ötlet:

Két szöveges változót használunk, az egyikben tároljuk a kitalálandó szót, a másikban az aktuálisan már kitalált karaktereket és a még hiányzó betűket és ez utóbbit módosítjuk az új tippel.

Megjegyzés:

A string típusú elemekben nem tudjuk kicserélni az i. karaktert, helyette konkatenációval újat készítünk. Mostantól a kódnak csak a feladathoz közvetlenül tartozó részeit közöljük!

Példa:

```
...
    class Program
    {
        static string inic( string kitalálandó)
        {
            return new String('-', kitalálandó.Length);
            //szövegtípus hosszaszor ismétlődik a '-' jel
        }
        static void következőtipp(string kitalálandó, ref string közbülső,
                                          ref int hibapont)
        {
            string segéd = ""; //itt építjük fel az új közbülső eredményt
```

```
        Console.Write("Melyik betűvel próbálkozik? ");
        char tipp = Convert.ToChar(Console.ReadLine());
        for (int i = 0; i < kitalálandó.Length; i++)
        {
            if (tipp == kitalálandó[i]) { segéd += tipp; } //konkatenáció
            else { segéd += közbülső[i]; }
        }
        if (közbülső == segéd) hibapont++;
        közbülső = segéd;
    }
    static void Main(string[] args)
    {
        string kitalálandó = "almafa", közbülső;
        char tipp;
        int hibapont = 0;
        közbülső = inic(kitalálandó); //annyi '-', ahány hosszú a kitalálandó
        while ( kitalálandó != közbülső && hibapont < 9 )
        {
            következőtipp(kitalálandó, ref közbülső, ref hibapont);
            Console.WriteLine(közbülső+" eddigi hibák száma:
                                        "+hibapont.ToString());
        }
        if (hibapont == 9) { Console.WriteLine("Vesztett"); }
        else { Console.WriteLine("Gratulálok nyert"); }
    }
```

Megjegyzés:

Ehhez a feladathoz később visszatérünk majd és megpróbáljuk megoldani a hibakezelését is!

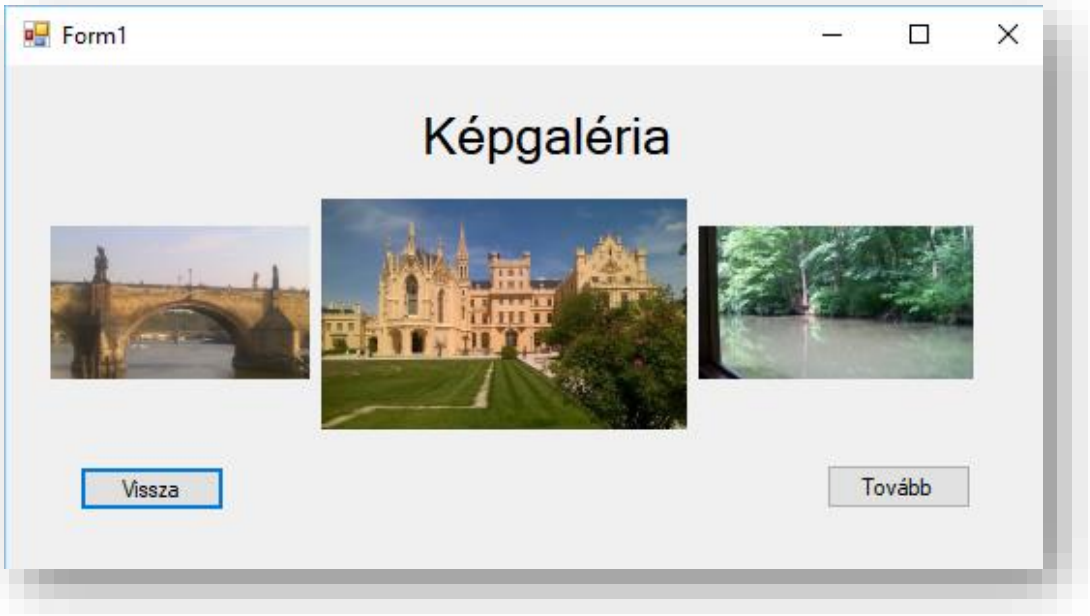
Nézzen utána!

Mit jelent a ref és az out kulcsszó a paraméterátadásnál?

Letölthető	
Akasztófa példa	https://gitlab.com/csharpttk/gyakorlat/Akasztofa

3. Feladat:

Ilyen egyszerű ismeretek birtokában is már készíthetünk látványos alkalmazást. Hozzunk létre egy nagyon egyszerű képnézegetőt a Windows platformon.



A megoldásban előforduló új nyelvi lehetőségek: Image, Button, Label vezérlő alkalmazása Windows Forms alkalmazásban.

Megoldási ötlet:

A képeket sorszámozzuk be. Egy előre és egy hátra lapozó gombbal az aktuálisan mutatandó képsorszámot változtatjuk és ennek megfelelő képet mutatjuk! Ehhez csak a két alapműveletre és a három operandusú operátorunkra van szükség!

Megjegyzés:

A képek Copy tulajdonságát állítsuk be always copy-ra!!

Példa:

```
private void btnVissza_Click(object sender, EventArgs e)
        {
           aktuális--;
           aktuális = aktuális < 1 ? 6 : aktuális;
            pBBal.ImageLocation = System.IO.Directory.GetCurrentDirectory() + "/Képek/" +
                                        (aktuális - 1 < 1 ? 6 : aktuális - 1).ToString() + ".jpeg";
            pBKözépső.ImageLocation = System.IO.Directory.GetCurrentDirectory() + "/Képek/" +
                                        (aktuális).ToString() + ".jpeg";
            pBJobb.ImageLocation = System.IO.Directory.GetCurrentDirectory() + "/Képek/" +
                                        (aktuális + 1 > 6 ? 1 : aktuális + 1).ToString() + ".jpeg";
        }
```

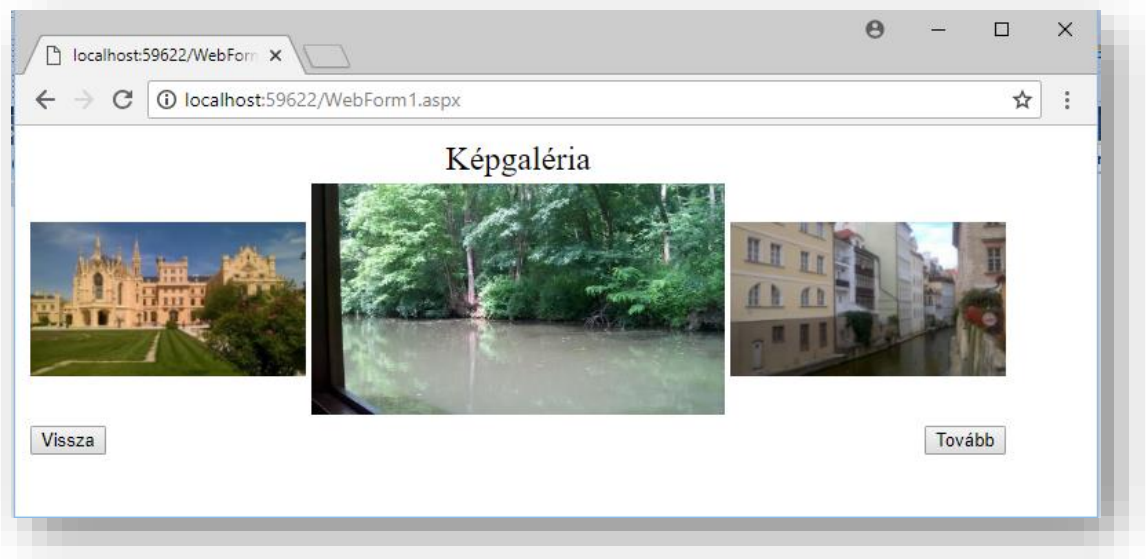
Letölthető	
Képnézegető	https://gitlab.com/csharpttk/gyakorlat/kepnezegeto

Megjegyzés:

A solution két projektet tartalmaz! Szükség esetén váltsunk a projektek között – jobb egérgomb, *Set as Startup Project* menüpont…

4. Feladat:

Ugyanezt az egyszerű képnézegetőt készítsük el webes alkalmazásként is.



A megoldásban előforduló új nyelvi lehetőségek: Image, Button, Label vezérlő alkalmazása Web Forms alkalmazásban. (Az ASP.Net kinézetre is nagyon hasonló, hiszen itt is ugyanolyan vezérlőkkel dolgozhatunk. Igaz, a vezérlők tulajdonságait másképp jegyzi a rendszer, szöveges formában!)

Példa, markup:

```
<div>
        <asp:Table ID="Table1" runat="server">
            <asp:TableRow runat="server">
                <asp:TableCell runat="server" columnspan="3">
                 <div style="margin-left: auto; margin-right: auto; text-align: center;">
                      <asp:Label ID="Label1" runat="server" Text="Képgaléria" Font-Size="X-Large"
                                                                  CssClass="StrongText"></asp:Label>
                 </div>
                </asp:TableCell>
            </asp:TableRow>
            <asp:TableRow runat="server">
                <asp:TableCell runat="server" >
                   <asp:Image ID="imgBal" runat="server" style="width:200px" ImageUrl="~/Képek/1.jpeg"/>
                 </asp:TableCell>
                <asp:TableCell runat="server" >
                    <asp:Image ID="imgKözépső" runat="server" style="width:300px"
                                                                  ImageUrl="~/Képek/2.jpeg"/>
```

```
                </asp:TableCell>
                …
            </asp:TableRow>
              <asp:TableRow runat="server">
                  <asp:TableCell runat="server" >
                      <asp:Button ID="btnVissza" runat="server" Text="Vissza"
                                                                  OnClick="btnVissza_Click" />
                  </asp:TableCell>
                   <asp:TableCell runat="server" >
                  </asp:TableCell>
                  …
              </asp:TableRow>
          </asp:Table>
      </div>
```

Példa, backend (mögöttes kód):

```
protected void btnTovább_Click(object sender, EventArgs e)
      {
          aktuális++;
          aktuális = aktuális > 6 ? 1 : aktuális;
          imgBal.ImageUrl = "Képek/" + (aktuális - 1 < 1 ? 6 : aktuális - 1).ToString() + ".jpeg";
          imgKözépső.ImageUrl = "Képek/" + (aktuális).ToString() + ".jpeg";
          imgJobb.ImageUrl = "Képek/" + (aktuális + 1 > 6 ? 1 : aktuális + 1).ToString() + ".jpeg";
      }
```

Megjegyzés:

A mögöttes kód viszont sz útvonal megadást tekintve azonos a Windows formos megoldáséval!

Megjegyzés:

A C# erejét mutatja, hogy kevés ismerettel is már különböző platformokra fejleszthetünk, ami a motivációt erősíatheti!

Letölthető	
Képnézegető	https://gitlab.com/csharpttk/gyakorlat/kepnezegeto

Megjegyzés:

A solution két projektet tartalmaz! Szükség esetén váltsunk a projektek között – jobb egérgomb, *Set as Startup Project* menüpont…

Önálló feladatok

1. Adjunk meg egy kiinduló szót konstansként, majd olvassunk be szavakat, amíg a felhasználó el nem rontja a szóláncot! Addig jó a szólánc, amíg az előző szó utolsó betűje megegyezik a következő szó első betűjével! Pl. alak, kati, ital, ló, ókor, …
2. Állapítsuk meg egy szóról, hogy palindrom-e, azaz visszafelé olvasva ugyanazt kapjuk-e! pl. sas, görög, abba, bab

3. Állapítsuk meg egy mondatról, hogy palindrom-e, azaz hagyjuk el a szóközöket és írásjeleket, ne tegyünk különbséget a kis és nagybetűk között és ugyanazt kapjuk-e visszafelé olvasva! Pl. Indul a görög aludni.

b) Összetett adattípusok

1. Feladat:

Csinosítsuk az akasztófa játékot (de még továbbra sem fogunk rajzolni)! Legyen egy szókészletünk, amiből kiválaszthatjuk a kitalálandó szót. Készítsünk egy C# nyelvű konzolos alkalmazást!

A megoldásban előforduló új nyelvi elemek: tömbök használata.

Megoldási ötlet:

Kis változtatással az előző verzióhoz képest, a kódba elhelyezett konstans kitalálandó szó helyett most egy szó sorozatból (tömbből) választjuk véletlenszerűen a feladatot.

Példa:

```
static void Main(string[] args)
 {
     string[] szavak = { "almafa", "programozás", "verzió", "tesztelés",
                                   "függvények", "ciklusok" };
     Random r = new Random(); int melyik = r.Next(szavak.Length);
     string kitalálandó = szavak[melyik], közbülső;
       ...
```

Nézzen utána!

Hogyan lehet statikus méretű tömböt deklarálni a nyelvben?

Letölthető	
Akasztófa-Tömb példa	https://gitlab.com/csharpttk/gyakorlat/Akasztofa_Tomb

2. Feladat:

A 2018 májusi emeltszintű érettségi feladat mostani tudásunkkal megoldható (elérhető *https://bit.ly/2oL8DkH*). A feladat eddigi tudásunkkal és a programozási tételeket ismerve már megoldható! Készítsünk egy C# nyelvű konzolos alkalmazást!

A megoldásban előforduló új nyelvi elemek: struktúrák használata.

Megoldási ötlet:

A tárolandó adatokat egy struktúrában helyezzük el, majd ebből építünk fel egy tömböt. A megoldáshoz a tételek mellett szükség lesz „józan paraszti észre” is, például arra, hogy nyilván azok maradnak bent a társalgóban a végén, akik páratlanszámszor haladtak át az ajtón vagy az lépett be utoljára, akit az adatokban hátulról keresve először megtalálunk.

Példa:

```
struct Tadat
```

```
   {
      public int mikor; //percekben
      public int ki;    //azonosító
      public bool állapot; //be => igaz
   }
class Program
 {
       const int maxáthaladás = 1000;
       const int maxszemély = 100;
       static int hányadat = 0;
       static Tadat[] társalgó = new Tadat[maxáthaladás];
       static int[] hányszormentát = new int[maxszemély + 1];
       static int személy;
       static void beolvasas()        {   string[] tmp = new string[3];
           string be = Console.ReadLine();
           while (be.Trim()!="")
           {
               tmp = be.Split(); //később fájlból
               társalgó[hányadat].mikor=60 * (Convert.ToInt32(tmp[0]) - 9) +
                                        Convert.ToInt32(tmp[1]);
               társalgó[hányadat].ki=Convert.ToInt32(tmp[2]);
               társalgó[hányadat++].állapot = tmp[3] == "be";
               be = Console.ReadLine();
           };
       }
             …
   }
```

Megjegyzés:

Ha nem akarunk fájlból olvasni, akkor sem szükséges kézzel begépelni az ajto.txt fájl tartalmát a teszteléshez! Kétféle megoldás közül választhatunk, az egyikben CTRL+C vel kimásoljuk a txt fájl tartalmát és futás közben a CTRL+v használatával bemásoljuk az adatokat, vagy parancsablakban átirányítást végzünk!

Megjegyzés:

Ehhez a feladathoz többször visszatérünk majd és megpróbáljuk majd az ügyesebb megoldás érdekében felhasználni a C# nyelv modern lehetőségeit !

Nézzen utána!

Mi a különbség a ++x és az x++ között?

Letölthető	
Érettségi feladat	https://gitlab.com/csharpttk/gyakorlat/Erettsegi_2018

Megjegyzés:

Ha tényleges fájlból közvetlenül akarunk adatot olvasni (ahogy az érettségin is kell!), annak több módja is van, de az a legegyszerűbb, ha egyszerre olvassuk be az összes adatot egy szövegtömbbe. Csak a beolvasás változik, de annak is csak egyetlen sora!

Megjegyzés:

Ha csak a fájlnévre akarunk hivatkozni, elhagyva a szükséges útvonal megadását, akkor ne felejtsük el beállítani a Property ablakban a Copy To Output directory tulajdonságot „Copy to allways" értékre!

Példa:

```
static void beolvasas()
{
    string[] tmp = new string[3];
    foreach ( string be in File.ReadAllLines("ajto.txt"))
        //beolvassa az egész fájlt egy szövegtömbbe
    {
            if (be.Trim() != "") //lehet a végén üres sor, később
                                 kivételkezeléssel dolgozunk majd
            {
                tmp = be.Split(); //
                társalgó[hányadik].mikor = 60 * (Convert.ToInt32(tmp[0]) - 9) +
                                                Convert.ToInt32(tmp[1]);
                társalgó[hányadik].ki = Convert.ToInt32(tmp[2]);
                társalgó[hányadik++].állapot = tmp[3] == "be";
            }
    };
}
```

Letölthető	
Érettségi feladat - fájlból	https://gitlab.com/csharpttk/gyakorlat/erettsegi2018_fajlbol

3. Feladat:

Valósítsuk meg N héten keresztül a lottó számhúzásokat és készítsünk statisztikát arról, hogy melyik számot hányszor húzták ki. Készítsünk egy C# nyelvű konzolos alkalmazást!

A megoldásban előforduló új nyelvi elemek: mátrix használata (tömbök tömbje).

Megoldási ötlet:

Figyelni kell arra, hogy azonos héten ugyanazt a számot nem húzhatják ki kétszer, így ezt folyamatosan ellenőrizni kell! Az egyes számok húzási gyakoriságához megszámolási tételt kell alkalmazni.

Példa:

```
…
static int[][] húzások;
static int hetekszáma;
```

```
static public void generálás()
{
    int újhúzás,k;
    Random r = new Random();
    húzások = new int[maxhét][];
    Console.Write("Hány hét? ");
    hetekszáma=Convert.ToInt16(Console.ReadLine()); //nincs ellenőrzés
    for (int i = 0; i < hetekszáma; i++)
    {
        húzások[i] = new int[húzásszám];
        int j = 0;
        while ( j < húzásszám) //amíg nincs 5 különböző szám
        {
            újhúzás = r.Next(1, maxérték + 1);
            k = 0;              //eldöntési tétel
            while (k < j && húzások[i][k] != újhúzás) k++;
                                    //ellenőrzés, volt-e már
            if (k == j)
            {
                húzások[i][j] = újhúzás;
                j++;
            }
        }
        for (int l=0;l<húzásszám;l++)
        {
            Console.Write("{0} ", húzások[i][l]);
        }
        Console.WriteLine();
         …
```

Gondolkodjon el!

> Hogyan lehetne ugyanezt a feladatot kétdimenziós tömbbel megoldani a most használt tömbök tömbje megoldás helyett?

Letölthető	
Érettségi feladat	https://gitlab.com/csharpttk/gyakorlat/Lotto

Megjegyzés:

Ez a megoldás csak klasszikus programozás nyelvi eszközöket használ az ismert programozási tételekre támaszkodva. A következő lépésben ezen kicsit módosítunk!

4. Feladat

Lottós példa kicsit ügyesebben, a modern eszközök felhasználásával.

A megoldásban előforduló új nyelvi elemek: Contains, Join.

Megoldási ötlet:

Az algoritmus változatlan marad, de vegyük észre, hogy például az előző verzió megoldásában szereplő eldöntési tétel helyett egy beépített nyelvi lehetőség is használható!

Példa:

```
…
            /*
        k = 0;
        while (k < j && húzások[i][k] != újhúzás) k++; //ellenőrzés, volt-e
        if (k == j) húzások[i][j] = újhúzás;
        */
        if (!húzások[i].Contains(újhúzás))
        {
            húzások[i][j++] = újhúzás;
        }
    }
    /*
    for (int l = 0; l < húzásszám; l++)
    {
        Console.Write("{0} ", húzások[i][l]);
    }
    Console.WriteLine();
    */
    Console.WriteLine(String.Join(" ", húzások[i]));
```

Megjegyzés:

A klasszikus megoldás az összehasonlítás kedvéért még megjegyzésekben szerepel a kódban!

Letölthető	
Érettségi feladat	https://gitlab.com/csharpttk/gyakorlat/Lotto_2

Önálló feladatok

1. Olvassunk be szavakat (egyelőre használjuk a fentebb leírt trükköt fájlból olvasás helyett) és tároljuk el ezeket egy tömbben! Adjuk meg a leghosszabb szót!
2. Olvassunk be szavakat (egyelőre használjuk a fentebb leírt trükköt fájlból olvasás helyett) és tároljuk el a szavakat egy tömbben! Adjuk meg, hogy szóláncot alkotnak-e (vagyis az utolsó szó utolsó betűjével kezdődik-e a következő szó).
3. Készítse el a 2017 évi emelt szintű informatika érettségi feladatait! Elérhető: https://bit.ly/2pGe1bP

III. Magasabb osztályba lépünk

Megjegyzés:

A feladatmegoldások során az első kötet (A C# nyelv lehetőségei) 4. és 5. fejezetének elméleti ismereteire fogunk támaszkodni.

a) Osztályok, öröklés

1. Feladat:

A 2018-as érettségi feladatot struktúra helyett oldjuk meg osztállyal!

A megoldásban előforduló új nyelvi elemek: osztály, objektum használata.

Megoldási ötlet:

Az algoritmus változatlan marad, csak az új nyelvi lehetőségeket kell beépíteni.

Példa:

```
class Tadat
{
    public int mikor { set; get; } //percekben
    public int ki { set; get; }    //azonosító
    public bool állapot { set; get; } //be => igaz
    public string óra_perccé_alakít()
    {
        return 9 + Convert.ToInt16( mikor / 60 ) + ":" + mikor % 60;
    }
}
class Program
{
    …
    static Tadat[] társalgó = new Tadat[maxáthaladás];
        …
    static void beolvasas()
    {
        …
        while (be.Trim() != "")
        {
            tmp = be.Split(); //később fájlból
            társalgó[hányadat] = new Tadat();
            …
        };
    }
        …
  static void mettőlmeddig()
```

```
    {
        Console.WriteLine("7. feladat");
        for (int i = 0; i < hányadat; i++)
        {
            if (társalgó[i].ki == személy)
            {
                if (társalgó[i].állapot)
                {
                    Console.Write(társalgó[i].óra_perccé_alakít() + "-");
                }
                else
                {
                    Console.WriteLine(társalgó[i].óra_perccé_alakít());
                }
            }
        }
        if (hányszormentát[személy] % 2 == 1) Console.WriteLine();
    }
```

Megjegyzés:

A Tadat osztály megadásánál property-ket és egyetlen függvényt készítettünk. Ennek az egyetlen függvénynek a segítségével átláthatóbbá tettük a kódunk óra:perc-óra:perc kiírását! **Korábban:** Console.WriteLine((9 + társalgó[i].mikor / 60) + ":" + társalgó[i].mikor % 60); **Most**: Console.Write(társalgó[i].óra_perccé_alakít());

Gondolkodjon el!

A struktúrákat tartalmazó tömbbel ellentétben itt az egyes tömbelemeket is példányosítani kellett. Vajon miért?

Nézzen utána!

Mi a feladata a konstruktornak és destruktornak? Nem készítettünk egyiket sem – akkor hogyan is tudtunk példányosítani?

Letölthető	
Érettségi feladat-osztály	https://gitlab.com/csharpttk/gyakorlat/Erettsegi_Osztaly

2. Feladat:

Készítsük el az akasztófa játék osztállyal történő megvalósítását!

A megoldásban előforduló új nyelvi elemek: osztályhoz operátor létrehozása.

Megoldási ötlet:

Az algoritmus lényegi része változatlan marad. Vegyük észre, hogy egy új tipp beolvasásával tulajdonképpen az akasztófa állapotán módosítottunk, azaz felfogható a megvalósítás operátorként is.

Példa:

```
public class Akasztófa_Alap
{
    public int hibapont { get; set; }
    public string kitalálandó { get; set; }
    public string közbülső { get; set; }
    private static string[] szavak = { "almafa", "programozás", "verzió", "tesztelés",
                                       "függvények", "ciklusok" };
    public Akasztófa_Alap() //konstruktor
    {
        Random r = new Random(); int melyik = r.Next(szavak.Length);
        kitalálandó = szavak[melyik];
        közbülső = new String('*', kitalálandó.Length);
        hibapont = 0;
    }
    public static Akasztófa_Alap operator +(Akasztófa_Alap állapot, char tipp)
    {   //új tipp érkezése megváltoztatja az állapotot
        Akasztófa_Alap újállapot = new Akasztófa_Alap()
        {
            hibapont = állapot.hibapont,
            kitalálandó = állapot.kitalálandó,
            közbülső = állapot.közbülső
        };
        string segéd = ""; //itt építjük fel az új közbülső eredményt
        for (int i = 0; i < újállapot.kitalálandó.Length; i++)
        {
            if (tipp == újállapot.kitalálandó[i]) { segéd += tipp; }//konkatenáció
            else { segéd += újállapot.közbülső[i]; }
        }
        if (újállapot.közbülső == segéd) újállapot.hibapont++;
        újállapot.közbülső = segéd;
        return újállapot;
    }
```

Megjegyzés:

Figyeljük meg az operátor paraméterezését!

Gondolja meg!

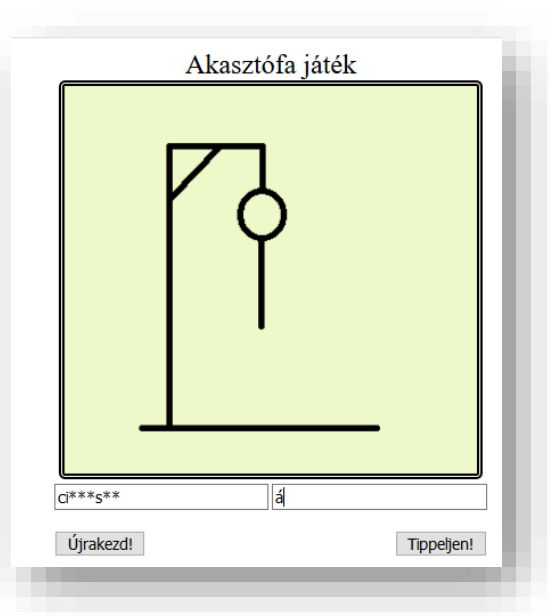
Felhasználva a mostani osztályunkat, hogyan tudna olyan játékosztályt készíteni, amelyik több egymás utáni játék impementálására szolgálna?

Letölthető	
Akasztófa – osztállyal	https://gitlab.com/csharpttk/gyakorlat/Akasztofa_Oszt

3. Feladat:

Készítsük el az akasztófa játék osztállyal történő megvalósítását most egy ASP.Net alkalmazásként!

A megoldásban előforduló új nyelvi lehetőség: leszármaztatás



Megoldási ötlet:

A megoldás lényegi része, az algoritmus változatlan marad, de a képi megjelenítés miatt az osztályunkat bővíteni kell egy új lehetőséggel (függvény), amely a hibapontoknak megfelelő kép azonosítót adja vissza. Ilyen esetben célszerű leszármaztatást készíteni.

Példa:

```
public partial class Akasztófa :
                    System.Web.UI.Page
  {
    private static Akasztófa_Leszármazott játék;
     private void inic()
     {
            játék = new Akasztófa_Leszármazott();
            Akasztófa_képe.ImageUrl = játék.megjelenítendőfájl();
            Közbülső_text.Text = játék.közbülső;
            …
     }
     protected void Page_Load(object sender, EventArgs e)
     {
         if (!IsPostBack)
         {
              inic();
         }
     }
     protected void Tipp_gomb_Click(object sender, EventArgs e)
     {
         if (Tipp_szöveg.Text.Length > 0) //írt-e valamit?
         {
             játék = (Akasztófa_Leszármazott)(játék + Tipp_szöveg.Text[0]);
             Akasztófa_képe.ImageUrl = játék.megjelenítendőfájl();
             Közbülső_text.Text = játék.közbülső;
             …
```

Megjegyzés:

Az osztályokat az áttekinthetőség kedvéért külön-külön fájlban hoztuk létre, de egyetlen fájlban is megadhattuk volna!

Megjegyzés:

Ha csak a fájlnévre akarunk hivatkozni, elhagyva a szükséges útvonal megadását, akkor ne felejtsük el beállítani a Property ablakban a Copy To Output directory tulajdonságot „Copy to allways” értékre!

Nézzen utána!

Mit is jelent a partial kulcsszó?

Gondolkodjon el!

Miért kellett alkalmazni a típuskényszerítést a következő sorban?
játék = (Akasztófa_Leszármazott)(játék + Tipp_szöveg.Text[0]);

Letölthető	
Akasztófa	https://gitlab.com/csharpttk/gyakorlat/ASP_Akasztofa

4. Feladat:

Készítsük el a számkitalálós játékunk ellenőrzött változatát, ahol a felhasználó nem tud rossz adatot megadni!

A megoldásban előforduló új nyelvi lehetőség: kivételkezelés

Megoldási ötlet:

A bevitt tippek esetében két fő hibalehetőség lehet. Az egyik, ha egész szám helyett valami mást ad meg a felhasználó, ilyenkor egy FormatException kivételt dob a rendszer. A másik esetben csak a feladat értelmezésének nem felel meg a bevitt adat, mert nem az 1,100 intervallumba tartozik. Ilyenkor nekünk magunknak kell szoftveres úton kivételt dobni!

Megjegyzés:

Dobhatunk tetszőleges beépített kivételt is, de készíthetünk egy sajátot az ős Exception osztály leszármazottjaként is!

Példa:

```
 Random r=new Random();
int szám = r.Next(1, 100);
Console.WriteLine("Gondoltam egy számot 1 és 100 között...(Kérem a tippeket)");
int tipp = -1;
while (tipp != szám)
{
    try
    {
        tipp = Convert.ToInt16(Console.ReadLine());
               //ha nem egész, FormatException típusú kivételt dob
        if (tipp < 1 || tipp > 100)
               throw new Egy_es_100Exception("Saját kivétel.. ");
         if (tipp < szám) { Console.WriteLine("Túl kicsi"); }
         if (tipp > szám) { Console.WriteLine("Túl nagy"); }
```

```
        }
         catch (Egy_es_100Exception ex)
         {
             Console.WriteLine(ex.Message+"Nem 1 és 100 közötti egész");
         }
         catch (FormatException ex)
         {
               Console.WriteLine("Nem egész");
         }
           Catch
           {
               Console.WriteLine("Egyéb hiba");
           }
         }
         Console.WriteLine("Gratulálok, eltalálta");
         …
class Egy_es_100Exception: Exception //saját kivételosztály
{
  public Egy_es_100Exception(string message):base(message)
  {
  }
 }
```

Megjegyzés:

A különböző kivétel típusokat a leszármazási fa hierarchiájának megfelelő sorrendben kezeli le. Emiatt az utolsó legyen a minden kivételt elkapó Exception típusú ág!

Nézzen utána!

Milyen típusú kivételt dob a rendszer, ha a megadott szám az ábrázolási tartományon kívül esik? Int16-ra konvertálunk, tehát 2 byteon ábrázolható értékre számítottunk…

Letölthető	
Számkitaláló	https://gitlab.com/csharpttk/gyakorlat/szamkitalalo_kivetel

5. Feladat:

Ismerjük csapatok közötti mérkőzések eredményeit. Rakjuk sorba ezeket a megszerzett összes gól szerint.

A megoldásban előforduló új nyelvi lehetőség: típussal paraméterezett függvények, interfész használat

Megoldási ötlet:

A rendezés algoritmusa bármelyiklehetne, most az egyszerű cserést választottuk. A feladat érdekessége az, hogy saját magunk által létrehozott *meccs* típusú elemek közötti összehasonlíthatóságot kell megoldani. Az összehasonlításhoz elő kell írnunk az ICompareable interfészt a CompareTo függvény megvalósításával.

Megjegyzés:

Olyan általános rendezési megoldást készítünk, amely több különböző típusra is használható. A letölthető mintakódban egészek illetve szöveges listákra is ellenőrizzük az elkészült kód alkalmazhatóságát.

Példa:

```
class meccs : IComparable  //tehát összehasonlíthatónak kell lennie az elemeknek
{
    public string kikjátszottak { set; get; }
    public int gólokelsőcsapat { set; get; }
    public int gólokmásodikcsapat { set; get; }
    public int CompareTo(object m)
    {
        meccs x = (meccs)m;
        if (gólokelsőcsapat + gólokmásodikcsapat > x.gólokelsőcsapat +
                                                   x.gólokmásodikcsapat)
            return -1;
        else if (gólokelsőcsapat + gólokmásodikcsapat == x.gólokelsőcsapat +
                                                   x.gólokmásodikcsapat)
            return 0;
        else return 1;
    }
    ...
}
static void cserésrendezés<ElemTípus>(List<ElemTípus> adatok)
                                                        where ElemTípus : IComparable
{   //ElemTípussal paraméterezett
    ElemTípus csere;
    for (int i = 0; i < adatok.Count - 1; i++)
        for (int j = i + 1; j < adatok.Count; j++)
        {
            if (adatok[i].CompareTo(adatok[j]) > 0)
                    //-1 kisebb, 0 egyenlő, 1 nagyobb
            {
                csere = adatok[i]; adatok[i] = adatok[j]; adatok[j] = csere;
            }
        }
}
...
static void Main(string[] args)
{
    ...
    List<meccs> meccsek = new List<meccs>(){
        new meccs(){kikjátszottak="a-b", gólokelsőcsapat=2,
                                          gólokmásodikcsapat=1},
```

```
            new meccs(){kikjátszottak="a-c", gólokelsőcsapat=1,
                                                gólokmásodikcsapat=1},
            new meccs(){kikjátszottak="b-c", gólokelsőcsapat=1,
                                                gólokmásodikcsapat=0},
        };
        ...
        Console.Write("Rendezetlenül: ");
        kiírás<meccs>(meccsek);
        cserésrendezés<meccs>(meccsek);
        Console.Write("Rendezetten: ");
        kiírás<meccs>(meccsek);
    }
```

Letölthető	
Rendezés-interfésszel	https://gitlab.com/csharpttk/gyakorlat/rendezes_interfesz

Önálló feladatok

- Készítse el a 2017 évi érettségi feladatot osztálytípus. felhasználásával
- Két dobókockával dobunk egyszerre egy társasjátékban. Egy osztályban tároljuk a dobások értékét és ismerünk egy N hosszú dobássorozatot. Számoljuk meg hányszor volt dupla hatos úgy, hogy az osztályunkhoz készítünk egy duplahatos logikai függvényt!
- Filmeket nézünk. Egy-egy film adatait (cím, hossz, nyelv) osztálytípusban ábrázoljuk. Adjuk meg a leghosszabb angol nyelvű filmet! Készítsünk leszármazottat, amelyik tartalmazza az első vetítési hét nézőszámát is.
- Az eddigi feladatokban valósítsa meg az ellenőrzött adatkezelést!

b) Generic

1. Feladat:

A 2018-as érettségi feladatot tömb helyett oldjuk meg típussal paraméterezett listával! Előnye, hogy dinamikusan növelhető a szükségnek megfelelően a mérete.

A megoldásban előforduló új nyelvi lehetőség: generic, lista típus.

Megoldási ötlet:

Az algoritmus változatlan marad, de a dinamikus lista használata miatt helytakarékosabb a megoldásunk, ha az eddigi maximális elemszám helyett aktuálisan kevesebb adattal dolgozunk!

Megjegyzés:

Ne felejtsük azért, hogy szükség van plusz adatok (mutatók) tárolására is, így ezek helyigénye is beleszámít a lefoglalt méretbe!

Példa:

```
class Program
{
    const int maxszemély = 100;
    static List<Tadat> társalgó = new List<Tadat>();
    static int[] hányszormentát = new int[maxszemély + 1];
    static int személy;
     static void beolvasas()
    {
         Tadat újadat;
        string[] tmp = new string[3];
        foreach (string be in File.ReadAllLines("ajto.txt"))
             //file meglétét nem ellenőrizzük, mert az érettséginél ez  nem elvárás
     {
            try
            {
                tmp = be.Split();
                        //lehet üres sor a fájl végén, ekkor hibát dobna a split
                újadat = new Tadat();
               újadat.mikor = 60 * (Convert.ToInt32(tmp[0]) - 9) +
                                       Convert.ToInt32(tmp[1]);
                újadat.ki = Convert.ToInt32(tmp[2]);
                újadat.állapot = tmp[3] == "be";
              társalgó.Add(újadat); //lista bővítése az új adattal
            }
           catch {; }
        };
    }
   static void első_utolsó()
    {
        int első = társalgó[0].ki;
        int utolsó = társalgó.Count - 1;
             //nem kellett megjegyezni hány adat van, lista hossza kiolvasható
        while (társalgó[utolsó].állapot) { utolsó--; }
        …
    }
```

Megjegyzés:

Mivel a lista hosszát lekérdezhetjük, a feladatban többé nincs szükség arra, hogy a beolvasás közben megszámoljuk hány adatot kaptunk! Ezzel is egyszerűsödött a feladat implementációja!

Megjegyzés:

Az érettségi feladatokat a valóságban sem kell adatellenőrzéssel ellátni, így mi sem tettük azt meg!

Nézzen utána!

Azt láttuk, hogy hogyan bővíthetjük a listát újabb elemekkel, de lehet belőle törölni is?

Letölthető	
Érettségi feladat-lista	https://gitlab.com/csharpttk/gyakorlat/Erettsegi_Lista

2. Feladat:

A lottós feladatot oldjuk meg generic használattal - listák listájával!

Megoldási ötlet:

A megoldás alapötlete, algoritmusa nem változik – legfeljebb az új lehetőség kínálta egyszerűbb megoldásokat kell végigvezetnünk!

Megjegyzés

A tömbök tömbje helyett használhatunk listák listáját. Elmondható, hogy a generic-ekben a paraméterezett típusok újabb generic-ek vagy akár tetszőleges más általunk létrehozott típusok is lehetnek!

Példa:

```
…
static List<List<int>> húzások=new List<List<int>>();
static int hetekszáma;
static public void generálás()
{
    …
    for (int i = 0; i < hetekszáma; i++)
    {
        húzások.Add( new List<int>() );
        while (húzások[i].Count < húzásszám)
        {
            újhúzás = r.Next(1, maxérték + 1);
            if (!húzások[i].Contains(újhúzás))
            {
                húzások[i].Add( újhúzás);
            }
        }
        húzások[i].Sort();
```

Megjegyzés:

A listáknál beépített metódusként használható a Sort( ), amellyel a típushoz tartozó alapértelmezett rendezés végrehajtható!

Nézzen utána!

Vajon egy tömb esetében is használható a Sort( ) metódus?

Letölthető	
Lottós, listával	https://gitlab.com/csharpttk/gyakorlat/Lotto_Lista

3. Feladat:

A változatosság kedvéért most egy WPF alkalmazást készítünk. Szófelhőnek nevezzük azt az ábrát, amelyen egy szövegben szereplő szavakat ábrázolunk. Minél gyakrabban fordul elő a szó, annál nagyobb méretben jelenítjük meg!

A megoldásban előforduló új nyelvi elemek: dictionary (szótár) típus használata

Megoldási ötlet:

Szógyakoriság megvalósításához adja magát a szótár (dictionary) típus használata, feleltessük meg a kulcsnak a szavakat és a kulcshoz (szóhoz) hozzátartozó érték pedig a gyakoriság legyen. A szöveget fel kell darabolni szavakra – ehhez rendelkezésre áll a Split() függvény, amely egy megadott elválasztó jel(ek) mentén darabol. (jellemzően ez, vagy ehhez hasonló függvény a programozási nyelvekben van). A szavakon végighaladva tulajdonképpen a megszámolás tételt alkalmazzuk megsegítve a szótár nyelvi lehetőséggel.

Megjegyzés:

Angol mondókát használunk, hiszen így elkerülhetjük azt a problémát, hogy magyarban egy szó különböző ragokkal is előfordul! A szótárunkban a szavak a kulcs szerepét töltik be, az érték pedig a szó előfordulásának száma.

Példa:

```
private List<string> vers=new List<string>() {
        "Six little ducks that I once knew," ,
        "Fat ones, skinny ones, fair ones too!",
        …
    };
Dictionary<string, int> szavakelőfordulása = new Dictionary<string, int>();
```

```
private void Feltölt(ref Dictionary<string, int> szavakelőfordulása)
{
    List<string> nemábrázolandó = new List<string>()
    { "a", "the", "this", "that", "and", "or","ones",… };
    string s = "";
    foreach  (string sor in vers)
        s += sor.ToLower();
    s = szöveg.Text.Replace('.', ' ').Replace('!', ' ').Replace('?', ' ');
    string[] szavak = s.Split(new char[] { ' ' });
    maxelőfordulás = 1;
    string kisbetűsszó;
    foreach (string szó in szavak)
    {
        kisbetűsszó = szó.Trim().ToLower();
         //csak kisbetűs változatot nézünk
        if (!nemábrázolandó.Contains(kisbetűsszó) && kisbetűsszó.Length>2)
        {
            if (!szavakelőfordulása.Keys.Contains(kisbetűsszó))
                szavakelőfordulása[kisbetűsszó] = 1;  //szó első előfordulása
            else
            {
                szavakelőfordulása[kisbetűsszó]++;
                if (maxelőfordulás < szavakelőfordulása[kisbetűsszó])
                        maxelőfordulás = szavakelőfordulása[kisbetűsszó];
            }
        }
    }
}  …
```

Megjegyzés:

Figyelünk arra, hogy az általános the, this, that stb. ne jelenjen meg a szófelhőben illetve az 1-2 betűs szavak se!

Megjegyzés:

A megjelenítésnél a szavakat tartalmazó TextBlock kontrollokat dinamikusan hozzuk létre és a szó gyakoriságának megfelelően három különböző betűmérettel készítjük el. Mind a kiírás színe, mind pedig a forgatása véletlenszámmal generált.

Gondolkodjon el!

Megoldható-e a feladat a Dictionary típus helyett SortedList segítségével?

Letölthető	
Szófelhő	https://gitlab.com/csharpttk/gyakorlat/Szofelho

Megjegyzés:

Mivel a nyelvbe beépített generic-ek igen használható tulajdonságokkal rendelkeznek és kényelmessé teszik a fejlesztő munkát, a továbbiakban ezeket fogjuk használni a feladatmegoldásokban!

Önálló feladatok

- Egy társasjátékban két kockával dobunk (a két kocka egy osztályként ábrázolt). A szabály alapján ha két hatost dobunk, újra mi jövünk. Ismerjük a dobások sorozatát – ami listában adott. Írjuk ki a dobásainkat!
- Készítsen el egy pakli magyar kártyát (32 lap, 4 szín, 8 figura) és keverje meg. Egy kártyalap legyen egy osztálytípusú (szín, figura), a pakli pedig legyen egy lista.
- Készítse el a 2017 évi emelt szintű informatika érettségi utolsó feladatát generic felhasználásával! Elérhető: https://bit.ly/2pGe1bP

IV. Speciális függvények

Megjegyzés:

A feladatmegoldások során az első kötet (A C# nyelv lehetőségei) 6. és 7. fejezetének elméleti ismereteire fogunk támaszkodni.

a) Delegáltak, névtelen metódusok, lambda kifejezések

a) Feladat:

A megszámolási, eldöntési és kiválogatási tételeket általánosan egy T tulajdonság függvénnyel fogalmazzuk meg a programozásaink során, hiszen a lényegi algoritmus a tényleges tulajdonságtól függetlenül leírható. Készítsünk egész értékeket tartalmazó listákon használható, tetszőleges tulajdonság függvénnyel működő általános implementációt!

A megoldásban előforduló új nyelvi elem: delegáltak, névtelen függvény, lambda függvény

Megoldási ötlet:

Használjuk fel az ismert algoritmusokat az implementációhoz, a T tulajdonság függvény helyett pedig használjunk delegáltat!

Példa:

```
delegate bool Ttul(int x);
 …
static int megszámol(List<int> lista, Ttul T)
 {
     int db = 0;
     foreach (int elem in lista)
         if (T(elem)) db++;
     return db;
 }
     …
 static bool kettőveloszthető(int x)
 {
     return x % 2 == 0;
 }
 static bool öttelosztható(int x)
 {
     return x % 5 ==0;
 }
 static bool hárommalosztható(int x)
 {
```

```
        return x % 3 == 0;
    }
    static void Main(string[] args)
    {
        List<int> lista = new List<int>() { 2,4,5,10,34,35,20,25};
        Console.WriteLine("5-tel oszthatóak száma: {0}",
                                megszámol(lista, ötteloszthatóe));
        Console.WriteLine("2-vel oszthatóak száma: {0}",
                                megszámol(lista, kettőveloszthatóe));
        Console.WriteLine("3-mal oszthatóak száma: {0}",
                                megszámol(lista, hárommaloszthatóe));
        …
```

Megjegyzés:

A delegált alatt függvény típust értsünk. Egy delegált típusú formális paraméter tetszőleges aktuális azonos szignatúrájú függvénnyel használható! (kettőveloszthatóe, hárommaloszthatóe, ötteloszthatóe – egy egész paraméterrel rendelkező, logikai értéket visszaadó függvénytípus)

Letölthető	
Tételek, delegált	https://gitlab.com/csharpttk/gyakorlat/delegalt

Megjegyzés

Variációk a delegált használatra és a beépített Func típusra.

Példa:

```
    static bool eldöntés(List<int> lista, Func<int, bool> T)
    //előre definiált Func, saját delegált helyett
     {
        int i = 0;
        while (i < lista.Count && !T(lista[i]))
            i++;
        return i < lista.Count;
     }
     …
     static void Main(string[] args)
     {
        …

        if (eldöntés(lista, kettőveloszthatóe)) //hagyományos függvény
            Console.WriteLine("Van köztük kettővel osztható");
        else
            Console.WriteLine("Nincs kettővel osztható");

        if (eldöntés(lista, delegate (int x) { return x % 3 == 0; }))
                                                    //névtelen függvény
```

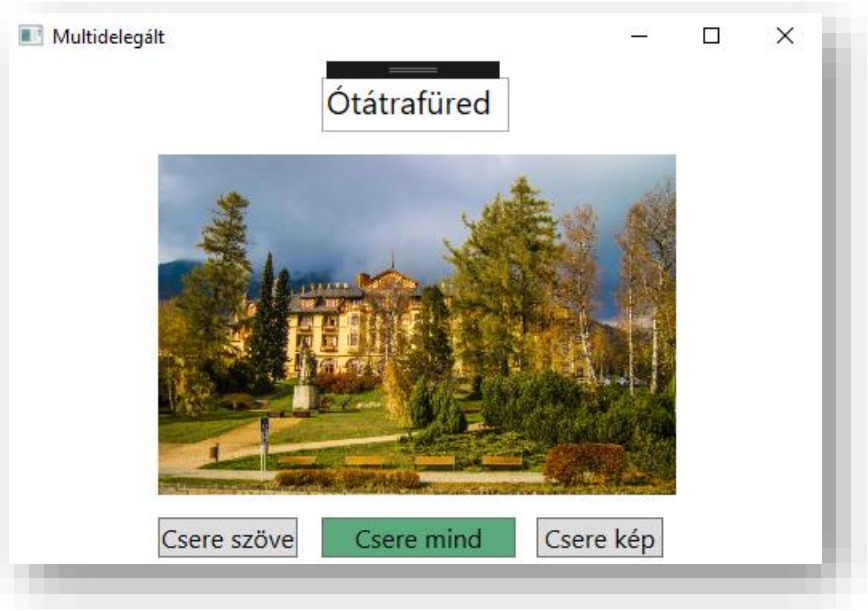
```
            Console.WriteLine("Van köztük hárommal osztható");
        Else
            Console.WriteLine("Nincs hárommal osztható");

        if (eldöntés(lista, x => x % 5 == 0)) //lambda kifejezés
            Console.WriteLine("Van köztük öttel osztható");
        else
            Console.WriteLine("Nincs öttel osztható");
```

Letölthető	
Tételek, delegált	https://gitlab.com/csharpttk/gyakorlat/delegalt_2

b) Feladat:

Készítsünk két képet és annak címét cserélgető alkalmazást, amely három gombbal működik! Az első gomb csak a szöveget, a középső a szöveget és a képet és az utolsó csak a képet cserélje meg!



A megoldásban előforduló új nyelvi elem: multidelegáltak

Megoldási ötlet:

Ha elkészítjük külön-külön a szöveg és a kép cseréjét végző eseménykezelőt, akkor a középső gomb click eseménykezelőjéhez mindkettőt hozzárendelhetjük!

Megjegyzés

Ne felejtsük el, hogy csak akkor használhatunk multidelegáltat, ha a függvénytípusnak nincs visszatérési értéke!.

Példa:

```
    private static int melyiksz = 0;
    private static int melyikk = 0;
```

```
public MainWindow()
{
    InitializeComponent();
    cserek.Click += képcsere; //csak képet cserél
    cseresz.Click += szövegcsere; //csak szöveget cserél
    cserem.Click += képcsere; //delegált függvény
    cserem.Click += szövegcsere; //második delegált
    //kattintásra mindkettő végrehajtódik
}
private void képcsere(object sender, RoutedEventArgs e)        {
    if (melyikk == 0)
        image.Source = new BitmapImage(new
                        Uri("lednice.jpg",UriKind.RelativeOrAbsolute));
    else
        image.Source = new BitmapImage(new Uri("tatra.jpg",
                                        UriKind.RelativeOrAbsolute));
    melyikk=(++melyikk) % 2;
}
private void szövegcsere(object sender, RoutedEventArgs e)
{
    if (melyiksz == 0)
        textBox.Text = "Lednice";
    else
        textBox.Text = "Ótátrafüred";
    melyiksz=(++melyiksz)%2;
}
```

Megjegyzés:

Ha csak a fájlnévre akarunk hivatkozni, elhagyva a szükséges útvonal megadását, akkor ne felejtsük el beállítani a Property ablakban a Copy To Output directory tulajdonságot „Copy to allways” értékre!

Gondolja meg:

Hol lehetne használni például egy memóriajáték megvalósításában (képeket megfordít és emlékezni kell, hol lehet a párja..) azt, hogy ugyanazt az eseménykezelőt több vezérlő eseményéhez is hozzá lehet rendelni? Milyen eseménykezelőt írna?

Letölthető	
Képcserélő, multidelegált	https://gitlab.com/csharpttk/gyakorlat/multidelegalt

Önálló feladatok

- Készítsünk az első feladathoz további Ttul típusú függvényeket. Például pozitív vagy negatív számok, prímek, négyzetszámok stb.
- Készítsünk el az első feladathoz tartozó függvények névtelen és lambda függvénykénti megadását!
- Készítsük egy WPF Tic-Tac-Toe játékot!

b) Függvény kiterjesztés

1. Feladat:

A magyar nyelvben sok az e betű. Viccesen eszperente nyelvnek nevezik, ha valaki csupa e betűt használ a mondanivalójában. Például: Mátyás hada: *Fegyveres emberek rettenetes fekete serege.*! Készítsünk alkalmazást, amelyik eldönti, hogy eszperente nyelvet használtünk-e.

A megoldásban előforduló új nyelvi elem: függvény kiterjesztés

Megoldási ötlet:

Gyűjtsük ki az összes magánhangzót, az e betű kivételével és vizsgáljuk, hogy a mondatunk tartalmazza-e valamelyiket!

Példa:

```
static class Eszperente
{
        public static bool eszperente_e(this string szöveg)
        {
            szöveg = szöveg.ToLower(); //ne kelljen külön nagy betűkre vizsgálni
            string mgh = "éiíuúöőüűaá"; //e kivételével a magánhangzók
            bool eszp = true;
            int i = 0;
            while (i<szöveg.Length && eszp)
            {
                eszp = eszp && !mgh.Contains(szöveg[i++]);
            }
            return eszp;
        }
    }
      …
      static void Main(string[] args)
        {
            Console.WriteLine("Kérek egy mondatot és megmondom eszperente-e...");
            string szöveg = Console.ReadLine();
            if (szöveg.eszperente_e())
                Console.WriteLine("Eszperente - csupa e");
            else
                Console.WriteLine("Nem eszperente...");
        }
```

Megjegyzés:

Több különböző függvény kiterjesztést is létrehozhat ugyanabban a statikus osztályban!

Gondolkodjon el!

Vajon mikor érdemes függvény kiterjesztést és mikor öröklést használni egy feladat megoldásánál?

Letölthető	
Eszperente	https://gitlab.com/csharpttk/gyakorlat/eszperente

2. Feladat:

Illik a születésnapost 8 napon belül felköszönteni! Ha ismerjük a családtagjaink születésnapját, Egy hónap, nap megadásával adjuk meg, hogy kinek van aznap a családban születésnapja! Egy adott dátum lekérdezésénél adja vissza az akkoriban felköszöntendőket!

A megoldásban előforduló új nyelvi elem: dátumtípus, kiterjesztés

Megoldási ötlet:

Egy szótárban tároljuk a születésnapokat. Két dátum típusú változó különbségét meghatározhatjuk, így a 8 napon belüliség könnyen megadható!

Példa:

```
public static List<string> ünnepelt(this DateTime ma)
 {   List<string> kik = new List<string>();
    try
     {
         string[] szn;
         DateTime szülinap;
         foreach (string datum in családiszületésnapok.Keys)
         {
             szn = datum.Split(new char[] { '.' });
             szülinap= new DateTime(ma.Year, Convert.ToInt16(szn[0]),
                                    Convert.ToInt16(szn[1]));
             if (Math.Abs(((szülinap - ma)).Days) < 8)
                kik.Add(családiszületésnapok[datum]);
             else
             {   //december-január közöttiek
                 szülinap = new DateTime(ma.Year + 1, Convert.ToInt16(szn[0]),
                                         Convert.ToInt16(szn[1]));
                 if (Math.Abs(((szülinap - ma)).Days) < 8)
                    kik.Add(családiszületésnapok[datum]);
                 else
                 {
                     szülinap = new DateTime(ma.Year - 1, Convert.ToInt16(szn[0]),
                                Convert.ToInt16(szn[1]));
                     if (Math.Abs(((szülinap - ma)).Days) < 8)
                        kik.Add(családiszületésnapok[datum]);
                 }
            }
         }
     }
     catch { ; };
     return kik;    Nézzen utána!
       ..
     if (dátum.ünnepelt().Count() > 0)
        {
```

```
            foreach (string ki in dátum.ünnepelt())
            {
                Console.Write("{0} ", ki);
            }
           Console.WriteLine();
        }
        else { Console.WriteLine("Nem volt senkinek 8 napon belül
                                  születésnapja"); }
```

Megjegyzés:

Nemcsak beépített típusokat (mint a példabeli string és dátum) lehet kiterjeszteni, hanem akár saját magunk által készített osztályokat is!

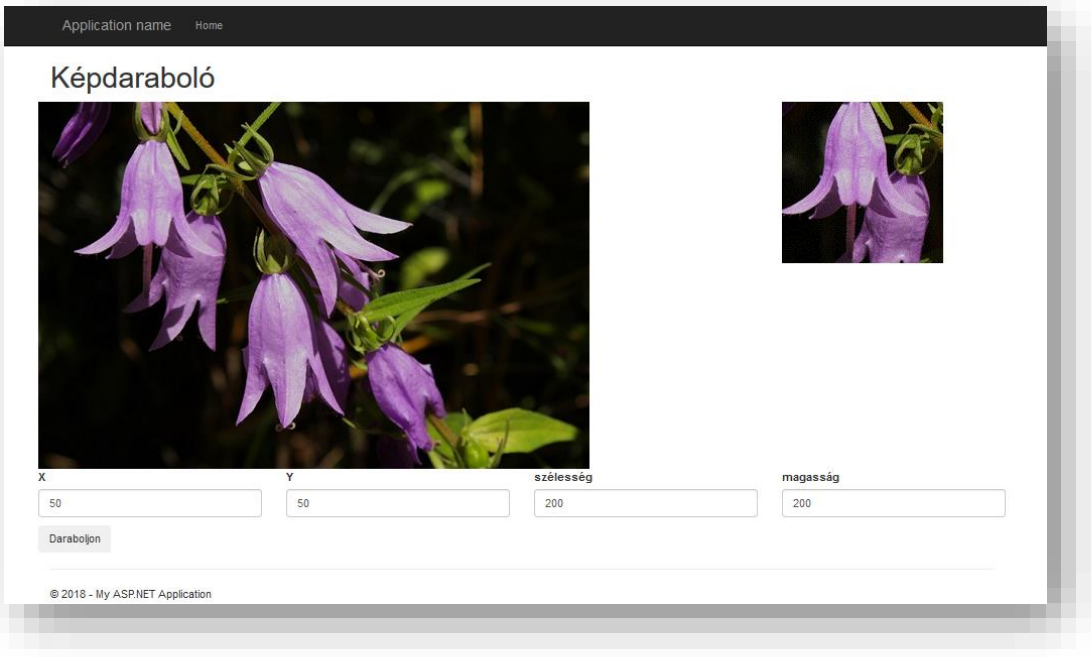
Gondolkodjon!

Hogyan lehetne még tárolni a családtagok születésnapját?

Letölthető	
Születésnap	https://gitlab.com/csharpttk/gyakorlat/szuletesnap

3. Feladat:

Képből szeretnénk egy téglalap alakú részt kivágni és azt megjeleníteni.



A megoldásban előforduló új nyelvi elem: függvény kiterjesztés paraméterekkel.

Megoldási ötlet:

Hozzunk létre egy kívánt méretű eredmény képet, majd pixelenként színezzük át az eredeti kép képpontjainak színére.

Példa:

```
static public string darabka(this Bitmap eredeti, int x , int y, int szélesség,
                                int magasság)
 {
     Bitmap képdarabka = new Bitmap(szélesség, magasság);
     for (int i = 0; i < szélesség; i++)
     {
         for (int j = 0; j < magasság; j++)
         {
             képdarabka.SetPixel(i, j, eredeti.GetPixel(x + i, y + j));
         }
     }
     MemoryStream ms = new MemoryStream();
     képdarabka.Save(ms, ImageFormat.Gif);
     var base64Data = Convert.ToBase64String(ms.ToArray());
     return "" + base64Data;
}
 // meghívása:
 …
 Képdarab.ImageUrl = eredeti.darabka(x, y, szélesség, magasság);
```

Megjegyzés:

Kicsit trükkösen kellett megadni a visszatérési értéket, mert az asp:Image kontroll csak egy ImageUrl tulajdonság megadásán keresztül kaphatja meg a megjelenítendő tartalmat.

Megjegyzés:

Ha csak a fájlnévre akarunk hivatkozni, elhagyva a szükséges útvonal megadását, akkor ne felejtsük el beállítani a Property ablakban a Copy To Output directory tulajdonságot „Copy to allways” értékre!

Letölthető	
Képdarabolás	https://gitlab.com/csharpttk/gyakorlat/Kepdarabolo

Önálló feladatok

- Készítsünk egy függvény kiterjesztést annak a feladatnak a megoldására, hogy egy egész számról meghatározzuk, hogy prím-e.
- Készítsünk a szöveg típushoz egy függvény kiterjesztést, amellyel azt adhatjuk meg, hogy palindrom-e (visszafele olvasva ugyanaz pl. sas, görög).
- Készítsünk a dátumtípushoz egy függvény kiterjesztést, amely azt adja vissza, hogy hét elejéről vagy végéről van-e szó. (hétfő vagy péntek)

V. Csipetnyi izgalom

Megjegyzés:

A feladatmegoldások során az első kötet (A C# nyelv lehetőségei) 8., 9. és 10. fejezeteinek elméleti ismereteire fogunk támaszkodni.

a) LINQ, Entity Framework

1. Feladat:

A programozási gyakorlatban igen sokszor a programozási tételeket használjuk fel a megoldásban. Nézzük meg, hogyan kell ezeket kódolni hagyományosan és milyen megoldásokat nyújt a LINQ! A feladatot egész elemeket tartalmazó listákra készítsük el!

A megoldásban előforduló új nyelvi lehetőség: LINQ

Megoldási ötlet:

A hagyományos megoldásokhoz az ismert algoritmusok kódolását végezzük el, a LINQ-s változatnál pedig a beépített lehetőségekre támaszkodunk!

Példa, Összegzési tétel hagyományosan:

```
static public int összegzés_hagyományosan()
{
     int s = 0;
     for (int i = 0; i < egyik.Count; i++)
          s += egyik[i];
     return s;
}
```

Példa, Összegzési tétel LINQ-val:

```
static public int összegzés_LINQ()
{
     return egyik.Sum();
}
```

Megjegyzés:

Később megnézzük majd azt is, hogy hogyan összegezünk egy összetett elemtípusból álló listát, sorozatot!!

Példa, Eldöntési tétel hagyományosan:

```
static public bool eldöntés_hagyományosan(Func<int,bool> T) //delegált használat
{
     int i= 0;
     while (i < egyik.Count && !T(egyik[i]))
          i++;
     return i<egyik.Count();
}
```

Példa, Eldöntési tétel LINQ-val:

```
static public bool eldöntés_LINQ(Func<int,bool> T)
{
    return  egyik.FindIndex(x=>T(x)==true)>=0; //-1, ha nincs
}
```

Megjegyzés:

A tulajdonságfüggvény típus megadását a paraméterben a beépített delegáltakkal Func< > oldottuk meg!

Példa, Megszámolási tétel hagyományosan:

```
static int megszámolás_hagyományosan(Func<int, bool> T)
{
     int db = 0;
     for(int i=0;i<egyik.Count;i++)
     {
          if (T(egyik[i])) db++;
     }
       return db;
   }
```

Példa, Megszámolási tétel LINQ-val:

```
static int megszámolás_LINQ (Func<int, bool> T)
{
   return  egyik.Where(x=>T(x)==true).Count();
}
```

Megjegyzés:

Figyeljük meg, hogy összetett kifejezések is felépíthetőek!! Először a where segítségével létrehozunk egy szűrt adatsorozatot, majd arra meghívjuk a count metódust!

Példa, Kiválasztási tétel hagyományosan:

```
static int kiválasztás_hagyományosan(Func<int, bool> T)
{
     int i = 0;
     while (!T(egyik[i]))
         i++;
    return egyik[i];
}
```

Példa, Kiválasztási tétel LINQ-val:

```
static int kiválasztás_LINQ(Func<int, bool> T)       //értéket adunk vissza
{
   return egyik.Where(x => T(x) == true).First();
}
```

Megjegyzés:

Ne felejtsük, hogy a where eredménye mindig egy adathalmaz, akkor is ha csak egyetlen eleme van!

Példa, Maximumkiválasztási tétel hagyományosan:

```
static int maximumkiválasztás_hagyományosan(Func<int, bool> T)
{
   int maxérték = egyik[0];
    for (int i=1;i<egyik.Count();i++)
    {
        if (maxérték < egyik[i])
              maxérték = egyik[i];
    }
    return maxérték;
}
```

Példa, Maximumkiválasztási tétel LINQ-val:

```
static int maximumkiválasztás_LINQ( Func<int, bool> T)
{
     return egyik.Max();
}
```

Megjegyzés.

Figyeljük meg, hogy mennyivel tömörebb írásmódot használhattunk a LINQ változatban!

Példa, Keresési tétel hagyományosan :

```
static (bool,int) keresés_hagyományosan(Func<int, bool> T) //tuple
{
     int i = 0;
     while (i < egyik.Count && !T(egyik[i]))
          i++;
     return (i < egyik.Count, egyik[i]);
 }
```

Példa, Keresési tétel LINQ-val

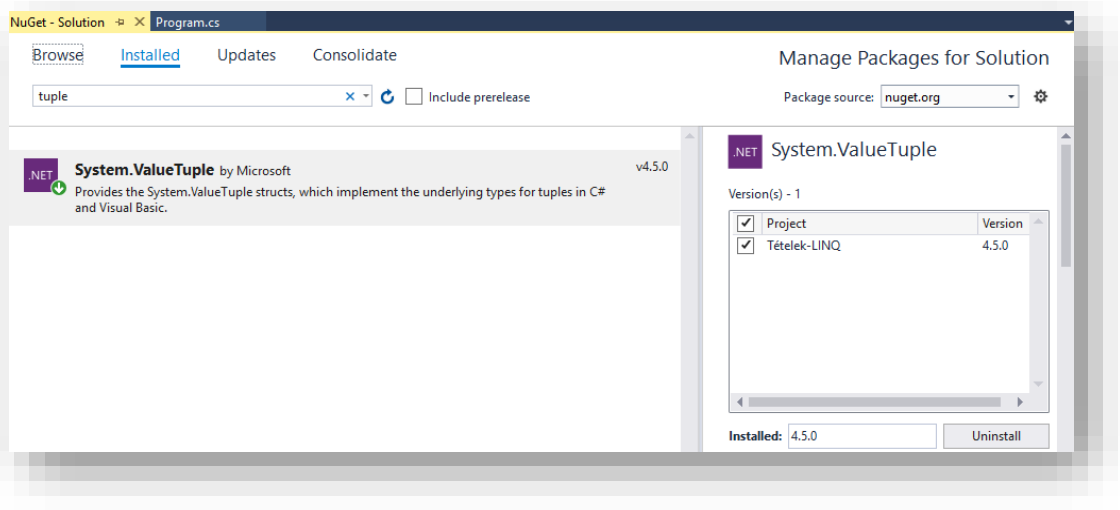
```
static (bool, int) keresés_LINQ(Func<int, bool> T) //tuple
{
   return egyik.Where(x => T(x) == true).Select(x => (egyik.IndexOf(x)> -1,  x )).First();
}
```

Megjegyzés:

A Select mindig egy sorozatot ad vissza, akkor is, ha csak egy elemből áll, így használnunk kell a First-öt!

Megjegyzés.

A függvény visszatérési értéke tuple! A jelenlegi Visual Studióban a használatához a System.ValueTuple package-t telepíteni kell! Tools/Nuget Package Manager/Manage Nuget Packages for Solution.



Példa, Kiválogatási tétel hagyományosan:

```
static List<int> kiválogatás_hagyományosan(Func<int, bool> T)
{
   List<int> eredmény = new List<int>();
   for (int i=0;i<egyik.Count();i++)
   {
         if (T(egyik[i]))
          {
              eredmény.Add(egyik[i]);
          }
    }
    return eredmény;
}
```

Példa, Kiválogatási tétel LINQ-val::

```
static List<int>  kiválogatás_LINQ(Func<int, bool> T)
{
    return egyik.Where(x => T(x) == true).ToList();
}
```

Megjegyzés.

Lista típussá alakíthattuk az eredményt!

Példa, Rendezési tétel hagyományosan:

```
  static public void rendez_hagyományosan()
{
     int csere;
     for (int i = 0; i < egyik.Count - 1; i++)
     {
         for (int j = i+1; j < egyik.Count; j++)
        {
            if (egyik[i] > egyik[j])
            {
```

```
                csere = egyik[i];
                egyik[i] = egyik[j];
                egyik[j] = csere;
            }
        }
    }
}
```

Példa, Rendezési tétel LINQ-val:

```
static public void rendez_LINQ()
{
      egyik.Sort(); //elemhez tartozó alapértelmezett reláció
}
```

Megjegyzés.

Figyeljük meg, hogy mennyivel tömörebb írásmódot használtunk a LINQ változatban!

Példa, Unió tétel hagyományosan:

```
 static public List<int> unio_hagyományosan()
{
    List<int> eredmény = egyik;                  //swallow copy
    for (int i = 0; i < másik.Count(); i++)
    {
        int j = 0;
        while (j < egyik.Count()-1 && egyik[j] != másik[i])
            j++;
        if (j >= egyik.Count())
            eredmény.Add(másik[i]);
     }
     return eredmény;
}
```

Példa, Unió tétel LINQ-val:

```
static public List<int> unio_LINQ()
{
      egyik.AddRange(másik);
      return egyik.Distinct().ToList();
}
```

Megjegyzés.

Természetesen nem válik szükségtelenné a tételek algoritmusának ismerete, de ennek a tudásnak a birtokában a LINQ használatával jelentősen felgyorsíthatjuk a kódolási munkát, ami például egy dolgozatban vagy akár az érettségin is hasznunkra lesz.!

Nézzen utána:

Melyik rendezési algoritmust használja a LINQ Sort() metódusa?

Letölthető	
Tételek-LINQ	https://gitlab.com/csharpttk/gyakorlat/tetelek_linq

2. Feladat:

Tételeink sorozatokon dolgoznak. Nézzük meg, hogy hogyan lehet kezelni kétdimenziós tömböket (C-szerű tömbök tömbjéről beszélünk most) LINQ segítségével. Szokványos feladat egy mátrix sorainak összegét, az összes elem összegét vagy maximum elemeket meghatározni.

A megoldásban előforduló új nyelvi elem: LINQ soronkénti feldolgozás

Megoldási ötlet:

Mindegyik részfeladatnál mind a sorokon, mind pedig a soron belüli elemeken végig kell haladni és elvégezni a szükséges máveleteket (összeadás vagy maximum kiválasztás). Az elv nem változik, de jóval hatékonyabb az eszközkészletünk!

Példa, soronkénti összeg

```
static public List<int> soronkénti_összeg()
{
        return mátrix.Select(sor=>sor.Sum()).ToList();
}
```

Megjegyzés:

Egymásba ágyazott számlálós ciklusok helyett egyetlen kifejezés!

Példa, elemek összege

```
 static public int elemek_összege()
  {
        return mátrix.Sum(sor => sor.Sum());
  }
```

Példa, sorok maximuma

```
static public List<int> soronkénti_max()
{
        return mátrix.Select(sor => sor.Max()).ToList();
}
```

Példa, elemek maximuma

```
static int elemek_maximuma()
{
        return mátrix.Max(sor => sor.Max());
}
```

Megjegyzés:

Láthatóan az alap feladatok megoldására két dimenziós tömbökön is gyors eszköz áll rendelkezésünkre!

Letölthető	
Mátrix-LINQ	https://gitlab.com/csharpttk/gyakorlat/matrix_linq

3. Feladat:

Látva a LINQ-s kifejezések hatékonyságát a tételek áttekintésénél, újra nézzük át az érettségi feladatunk megoldását!

A megoldásban előforduló új nyelvi elem: LINQ (Distinct())

Megoldási ötlet:

A korábbi algoritmusok átfogalmazását végezzük csak el.

Példa, beolvasás hagyományosan és LINQ-val:

```
static void beolvasas()
 {
     Tadat újadat;
     string[] tmp = new string[3];
     foreach (string be in
           File.ReadAllLines("ajto.txt"))
     {
         try
         {
           tmp = be.Split(); //ha üres az utolsó sor, hibát dobna
           újadat = new Tadat();
           újadat.mikor = 60 *  Convert.ToInt32(tmp[0]) - 9)  +   Convert.ToInt32(tmp[1]);
           újadat.ki = Convert.ToInt32(tmp[2]);
           újadat.állapot = tmp[3] == "be";
           társalgó.Add(újadat); //lista  bővítése az új adattal
          }
          catch {; }
       };
  }
```

Példa, beolvasás LINQ-val:

```
static void beolvasas(){
     társalgó=File.ReadAllLines("ajto.txt")
                  .Where(s=>s.Trim()!="")
                 .Select(sor => new Tadat {
                    mikor=60*(Convert.ToInt16(sor.Split()[0])-9)+Convert.ToInt16(sor.Split()[1]),
                    ki = Convert.ToInt16(sor.Split()[2]),
                    állapot = sor.Split()[3] == "be"})
                . ToList();
}
```

Megjegyzés

A Where(s=>Trim()!="") kiszűri az esetleges üres sorokat. A mikor azonnali összeállítása miatt nincs szükség az újadat változó deklarációjára. A ToList() pedig feleslegessé teszi a lista kézi feltöltését az Add( ) használatát!

Példa, első-utolsó függvény hagyományosan:

```
static void első_utolsó()
{
    int első = társalgó[0].ki;
    int utolsó = társalgó.Count - 1;
     //nem kellett megjegyezni hány adat van, lista hossza kiolvasható
     while (társalgó[utolsó].állapot)
       {
            utolsó--;
       }
         …
}
```

Példa, első-utolsó függvény LINQ-val:

```
static void első_utolsó()
{
    int első = társalgó[0].ki;
    int utolsó = társalgó
       .Where(x => x.állapot ==false).Last().ki;
    …
}
```

Megjegyzés

Kiválasztási tétel:

Példa, ki hányszor ment át függvény hagyományosan:

```
static void kihanyszor()
{
 for (int sorszám = 0; sorszám < társalgó.Count; sorszám++)
 {
    hányszormentát[társalgó[sorszám].ki]++;
 }
  Console.WriteLine("3. feladat");
  for (int i = 1; i <= maxszemély; i++)
  {
   if (hányszormentát[i] != 0)
        Console.WriteLine("{0} {1}", i, hányszormentát[i]);
 }
}
```

Példa, ki hányszor ment át függvény LINQ-val:

```
static void kihanyszor()
  {
```

```
      hányszormentát=társalgó.GroupBy(x => x.ki)
                            .Select(x => new Thányszor{ ki = x.Key, hányszor = x.Count() })
                            .OrderBy(x=>x.ki)
                             .ToList();
       Console.WriteLine("3. feladat");
       foreach (var személy in hányszormentát)
      {
          Console.WriteLine("{0} {1}", személy.ki, személy.hányszor);
      }
}
```

Megjegyzés

Csoportosítjuk személyek szerint, majd megszámoljuk az előfordulásukat!

Példa, bentmaradtak a végén függvény hagyományosan :

```
static void bentmaradtak()
 {
  Console.WriteLine("4. feladat");
    Console.Write("A végén a társalgóban voltak:");
    for (int i = 1; i <= maxszemély; i++)
     {
       if (hányszormentát[i] % 2 == 1) //többször ment be, mint ki, tehát bent van
         Console.Write(i.ToString() + " "); //később fileba
      }
      Console.WriteLine();
}
```

Példa, bentmaradtak a végén függvény LINQ-val:

```
static void bentmaradtak()
{
    Console.WriteLine("4. feladat");
    Console.Write("A végén a társalgóban voltak:");
    foreach  (Thányszor személy in hányszormentát)
    {
      if (személy.hányszor % 2 == 1)  //többször ment be, mint ki,  tehát bent van
      Console.Write(személy.ki + " ");  //később fileba
    }
    Console.WriteLine();
}
```

Példa, legtöbben függvény hagyományosan:

Megjegyzés:

Ha valaki belép, eggyel többen lesznek, ha kimegy, egyel kevesebben az előző perchez képest!.

```
static void legtobben()
{
  int maxmikor = társalgó[0].mikor;    //egy ember legalább van bent
  int[] éppenhányanvannakbent = new int[6 *60 + 1];
```

```
 éppenhányanvannakbent[társalgó[0].mikor] = 1; //az első ember biztos befelé megy
 for (int i = társalgó[0].mikor + 1; i < társalgó.Count; i++)
 {
    if (társalgó[i - 1].mikor !=  társalgó[i].mikor)
    {
           éppenhányanvannakbent[társalgó[i].mikor] =
           éppenhányanvannakbent[társalgó[i - 1].mikor];
    }
    if (társalgó[i].állapot)
    {
            ++éppenhányanvannakbent[társalgó[i].mikor];    //jött még valaki
            if (éppenhányanvannakbent[maxmikor] <  éppenhányanvannakbent[társalgó[i].mikor])
                maxmikor = társalgó[i].mikor;
    }
    else
    {
         --
           éppenhányanvannakbent[társalgó[i].mikor];   //valaki elment
   }
}
    Console.WriteLine("5. feladat");
    Console.WriteLine("Például {0}-kor voltak legtöbben a teremben", (9 + maxmikor / 60) + ":" +
                                                                      maxmikor % 60);
}
```

Példa, legtöbben függvény LINQ-val:

```
static void legtobben()
 {
    List<int> időpontok = társalgó
                          .Select(x => x.mikor)
                          .Distinct().ToList();
     int bentlevők,maxbentlevők = -1, maxidő = -1;
     foreach (int eddig in időpontok)
    {
        bentlevők = társalgó
                    .Where(x => x.mikor <= eddig)
                .Where(xx => xx.állapot == true)
                     .Count() - társalgó.Where(x => x.mikor <= eddig)
                     .Where(X => X.állapot == false)
                .Count();
         if ( maxbentlevők < bentlevők)
        { maxbentlevők = bentlevők;
            maxidő = eddig;
        }
     }
   Console.WriteLine("5. feladat");
   Console.WriteLine("Például {0}-kor voltak legtöbben a teremben",(9 + maxidő / 60) + ":"
                                                                    + maxidő % 60);
}
```

Megjegyzés:

> Az időpontokat kigyűjtjük. Szűrjük az egyes időpontokra az adatokat és ebben számoljuk a belépéseket, kilépéseket. Az alapgondolat nem változott!

Példa, mettől, meddig függvény hagyományosan:

```
static void mettőlmeddig()
{
    Console.WriteLine("7. feladat");
    for (int i = 0; i < társalgó.Count; i++)
   {
       if (társalgó[i].ki == személy)
       {
        if (társalgó[i].állapot)
        {
         Console.Write( társalgó[i].óra_perccé_alakít()+ "-");
         }
           else
         {
             Console.WriteLine( társalgó[i].óra_perccé_alakít());
           }
        }
     }
    if (hányszormentát[személy] % 2 == 1)
           Console.WriteLine();
}
```

Példa, mettől, meddig függvény LINQ-val:

```
static void mettőlmeddig()
  {
      Console.WriteLine("7. feladat");
    foreach (var sz in társalgó.Where(x => x.ki == személy))
    {
        if (sz.állapot)
        {
                Console.Write(sz.óra_perccé_alakít() + "-");
        }
        else
        {
                Console.WriteLine(sz.óra_perccé_alakít());
        }
    }
    if (társalgó.Count(x=>x.ki==személy) % 2 == 1)
         Console.WriteLine();
  }
```

Megjegyzés:

> Ebben a részfeladatban csak a társalgó sorozat bejárását szűkítjük egy szűrővel – egyébként változatlan a megoldás!

Feladat

A már ismert Lotto-s feladatunk megoldásában is használjuk fel a LINQ lehetőségeit!

A megoldásban előforduló új nyelvi elemek: LINQ (SelectMany, GroupBy, OrderBy)

Megoldási ötlet:

Transzformáljuk az összes heti 5 számot tartalmazó húzáslistát (két dimenziós lista) egyetlen listává (egy dimenzióssá), ami már egyszerűbben kezelhető!

Példa, statisztika függvény hagyományosan:

```
static public void statisztika()
{
  int[] hányszor = new int[maxérték + 1]
  for (int i = 1; i <= maxérték; i++)
         hányszor[i] = 0;
  for (int j = 0; j < hetekszáma; j++)
  {
    for (int k = 0; k < húzásszám; k++)
    {
       ++hányszor[húzások[j][k]];
    }
 }
 for (int i = 1; i <= maxérték; i++)
{
    Console.WriteLine("{0} - {1} ", i,hányszor[i]);
  }
}
```

Példa, statisztika függvény LINQ-val:

```
static public void statisztika()
 {
   var eredmény = húzások
                 .SelectMany(x => x)
                 .GroupBy(x=>x)
                 .Select(y=>new { szám = y.Key, gyakoriság = y.Count() })
                 .OrderBy(r=>r.szám);
  foreach (var sz in eredmény)
  {
      Console.WriteLine("{0} - {1} ", sz.szám,
                                  sz.gyakoriság);
   }
 }
```

Megjegyzés:

A SelectMany segítségével egyetlen sorozattá alakítjuk a listák listáját. A GroupBy segítségével a kihúzott számokra csoportosítunk. A Select használatával elkészítjük az eredményt, az egyes számok húzási gyakoriságát a Count-tal. Ahhoz, hogy a számok növekvő sorrendben jelenjenek meg, az OrderBy-t használjuk!

Letölthető	
Lotto-LINQ	https://gitlab.com/csharpttk/gyakorlat/lotto_linq

Önálló feladatok

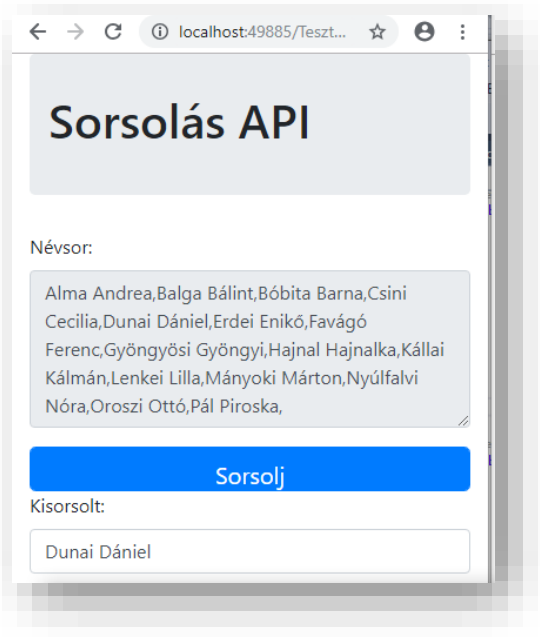
- Készítsük el azt a LINQ-s kifejezést, amelyik megszámolja egy számlistában előforduló maximális elemeket!
- Készítsük el azt a LINQ kifejezést, amelyik egy tömbök tömbjében visszaadja az átlóban található értékeket lista formájában!
- Készítse el a 2017 évi emelt szintű informatika érettségi feladatát LINQ-s kifejezések felhasználásával! Elérhető: https://bit.ly/2pGe1bP

b) Web API hívás, kitekintés

Végezetül ízelítőként megmutatjuk, hogy hogyan lehet Web API-t készíteni a Visual Stúdió 2017 és a C# nyelv segítségével. Megnézzük, hogyan lehet ezt egy html oldalról vagy akár konzol alkalmazásból elérni. Érdemes kipróbálni akár az ingyenes Azure azonosítóval is és publikálni az API-t a felhőbe is!

1. Feladat:

Feleltetni szeretnénk és a névsorból véletlenszerűen szeretnénk valakit kiválasztani. Készítsünk ehhez egy Web API-t, amelyik tárolja a névsort és két lekérést tesz lehetővé: a teljes névsort és egy véletlenül kiválasztott nevet! A megoldás tartalmazza az API tesztelési lehetőségét is!



A megoldásban előforduló új nyelvi lehetőség: Web API használat

Megoldási ötlet:

Listából véletlenül kiválasztott elem már ismert. Algoritmus szinten a feladat nem tartalmaz újdonságot!

Megjegyzés:

Hozzunk létre egy új projektet, a Web Application (Asp Web Application (.Net Framework) sablon alapján. Majd válasszuk az Empty sablont és jelöljük be a WebAPI-t. A Solution Explorerben a Controllers mappához adjunk egy Web APi controllers SorsolasController néven.

Megjegyzés:

Az API-ban a Get kéréseket valósítottuk csak meg. A GetAll( ) visszadja kérésre a teljes névsor listát, a GetKiválasztott(int id) pedig egy véletlenül meghatározott nevet. Mivel a Get kéréseket a rendszer a paraméterek száma és típusa alapján azonosítja, ebban az esetben az id szerepe csak arra korlátozódik, hogy megkülönböztesse egymástól a hívásokat! Részletesebben olvashat erről a https://bit.ly/2nTZchN címen!

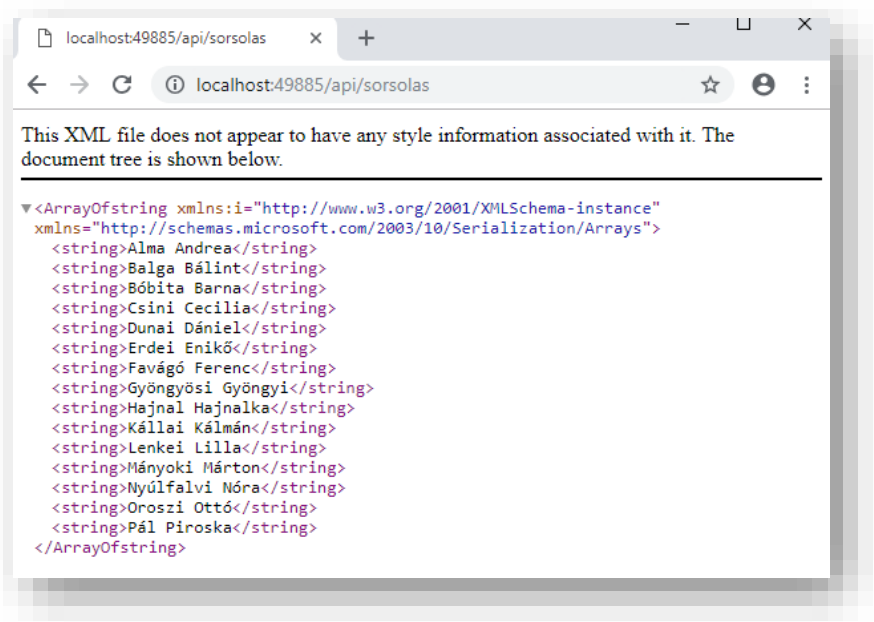
Példa, SorsolasController:

```
public class SorsolasController : ApiController
 {  public static List<string> nevek = new List<string>(){"Alma Andrea", "Balga Bálint",
        "Bóbita Barna","Csini Cecilia", "Dunai Dániel", "Erdei Enikő","Favágó Ferenc",
        "Gyöngyösi Gyöngyi","Hajnal Hajnalka","Kállai Kálmán","Lenkei Lilla","Mányoki Márton",
        "Nyúlfalvi Nóra","Oroszi Ottó","Pál Piroska" };
    [EnableCors(origins:"*", headers: "*", methods: "*")]
    public IHttpActionResult GetAll() //id paraméternév a default!
    {
       try
        {
               return Ok(nevek);
        }
        catch
        {
            return NotFound();
        }
    }
    [EnableCors(origins: "*", headers: "*", methods: "*")]
    public IHttpActionResult GetKivalasztott(int id)
                      //id-t nem használom - paraméter csak megkülönbözteti a hívásokat
    {
        Try
        {
            Random r = new Random();
            return Ok(nevek[r.Next(nevek.Count)]); //véletlenszerűen választott név
        } catch {
            return NotFound();
        }
     }
 }
```

Megjegyzés:

Ahhoz, hogy másik alkalmazásból is meghívható legyen a Web API-nk, szükség van az *[EnableCors(origins: "*", headers: "*", methods: "*")]* attribútumra! Ehhez a *cors* nuget package-t telepíteni kell! Szükség van még ennek engedélyezésére is a WebApi.config fájlban a *config.EnableCors();* utasítással! Mindezek megtalálja a letölthető anyagban!

A WebApi kész is van, indítsuk el az alkalmazást és teszteljük a böngészőnkben (az url-hez adjuk hozzá az api/sorsolas útvonalat). Máris láthatjuk az eredményt, a névsort:



Készítsünk hozzá egy teszt.html oldalt, ahonnan AJAX hívással meghívjuk az API-t.

Példa, teszt.html oldal:

```
<div class="container">
        <div class="jumbotron">
            <h1 > Sorsolás API</h1>
        </div>
        <div class="form-group">
            <label for="nevsor">Névsor:</label>
            <textarea readonly class="form-control"  rows="5" id="nevsor"></textarea>
        </div>
        <input type="button" value="Sorsolj" class="form-control btn btn-primary btn-lg"
                                                                        onclick="sorsolj();" />
            <div class="form-group">
                <label for="kiválasztott">Kisorsolt:</label>
                <p id="kivalasztott" class="form-control" />
            </div>
        </div>
        <script src="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.0.3.min.js"></script>
        <script>
            var uri = 'api/sorsolas';
            $(document).ready(function () {
                $.getJSON(uri)
                    .done(function (data) {
                        $.each(data, function (i, item) {
                            var nev = item+","
                            $('#nevsor').text($('#nevsor').text()+nev);
                        }); //End of foreach Loop
```

```
            })
            .fail(function (jqXHR, textStatus, err) {
                $('#kivalasztott').text('Hiba: ' + err);
            })
    })
    function sorsolj() {
        var id = 0;
        $.getJSON(uri + '/' + id)
            .done(function (data) {
                $('#kivalasztott').text(data);
            })
            .fail(function (jqXHR, textStatus, err) {
                $('#kivalasztott').text('Hiba: ' + err);
            });
    }
</script>
```

Megjegyzés:

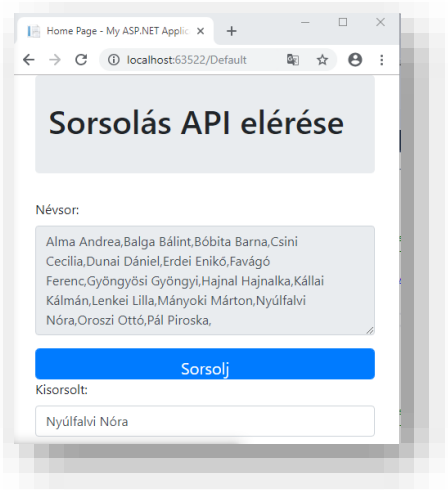
Ezt futtava megkapjuk a feladat leírásban mutatott képernyőképet!

Letölthető	
SorsolasWAPI	https://gitlab.com/csharpttk/gyakorlat/sorsolaswebapi

2. Feladat:

Az előbbi Web API alkalmazásunkat hívjuk meg egy ettől függetlenül létrehozott Web Form alkalmazásból!

A megoldásban előforduló új nyelvi lehetőség: Web API meghívása másik webes alkalmazásból AJAX hívással!



Megjegyzés:

Hozzunk létre egy új projektet, a Web Form Application sablon alapján. Töröljük a most felesleges About és Contact formokat és javítsuk a Site.mastert ennek megfelelően.

Megjegyzés:

A meghívás majdnem ugyanaz, mint az teszt.htm-ben volt, csak a teljes elérési utat meg kell adnunk! Ehhez javítsuk ki az *uri*-t a saját böngészőnkben látható útvonalra!

Példa, Default.aspx részlete:

```
<script>
```

```
        //var uri = ide írjuk az  api címét;
        var uri = "http://localhost:49885/api/sorsolas";
        $(document).ready(function () {
            $.getJSON(uri)
                .done(function (data) {
                    $.each(data, function (i, item)
                       var nev = item + ","
                        $('#nevsor').text($('#nevsor').text() + nev);
                    }); //End of foreach Loop
                })
                .fail(function (jqXHR, textStatus, err) {
                    $('#kivalasztott').text('Hiba: ' + err);
                })
        })
        …
    </script>
```

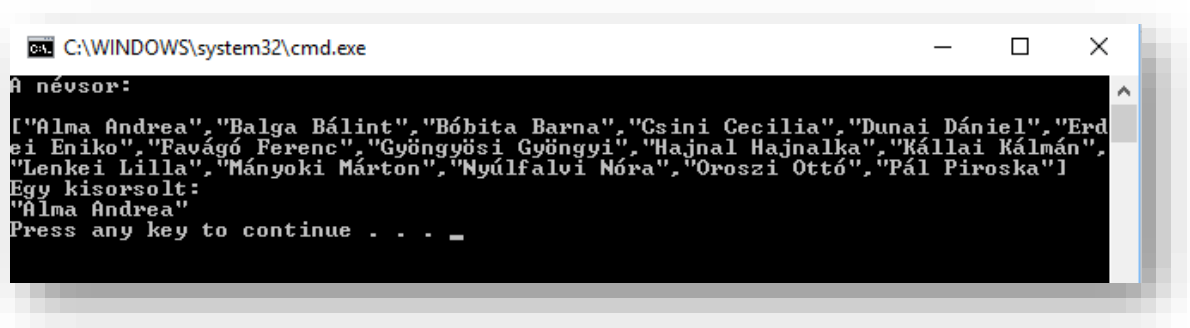
Megjegyzés:

Futtassuk az alkalmazásunkat, de ne feledkezzünk meg arról, hogy az API-nak is futnia kell ugyanakkor! Az eredmény a feladat kitűzésénél látható!

Letölthető	
SorsolasWebApp	https://gitlab.com/csharpttk/gyakorlat/sorsolaswebapp

3. Feladat:

Az előbbi Web API alkalmazásunkat hívjuk meg egy ettől függetlenül létrehozott Console alkalmazásból!



A megoldásban előforduló új nyelvi lehetőség: Web API meghívása console alkalmazásból!

Megjegyzés:

Hozzunk létre egy új Console projektet és adjunk hozzá egy osztályt, amelyben szinkron hívást valósítunk meg!

Példa, Sorsolás osztály

```
class Sorsolas
    {
        public void GetAll() //az összes név
        {
            using (var client = new WebClient()) //WebClient
           {
                client.Encoding = Encoding.UTF8; //magyar ékezetes betűk elérése
                client.Headers.Add("Content-Type:application/json"); //Content-Type
                client.Headers.Add("Accept:application/json");
                var result = client.DownloadString("http://localhost:49885/api/sorsolas");
                                        //URI módosítsuk a saját gépen vagy Azure-ban futóra
                Console.WriteLine(Environment.NewLine + result);
            }
        }
        public void GetKiválasztott(int id) //egy kisorsolt
        {
            using (var client = new WebClient()) //WebClient
            {
                client.Encoding = Encoding.UTF8; //magyar ékezetes betűk elérése
                client.Headers.Add("Content-Type:application/json"); //Content-Type
                client.Headers.Add("Accept:application/json");
                var result = client.DownloadString("http://localhost:49885/api/sorsolas/" + id.ToString());
                                          //URI módosítsuk a saját gépen  vagy  Azure-banfutóra
                Console.WriteLine(result);
            }
        }
    }
```

Megjegyzés:

Létre kell hoznunk egy WebClient objektumot, ami az Api hívást kezeli és azon keresztül meghívni azt! A main függvényben pedig példányosítani kell az osztályunkat és végrehajtani a hívást!

Megjegyzés:

Az uri címet írjuk át a sajátunkra!

Példa, main függvény:

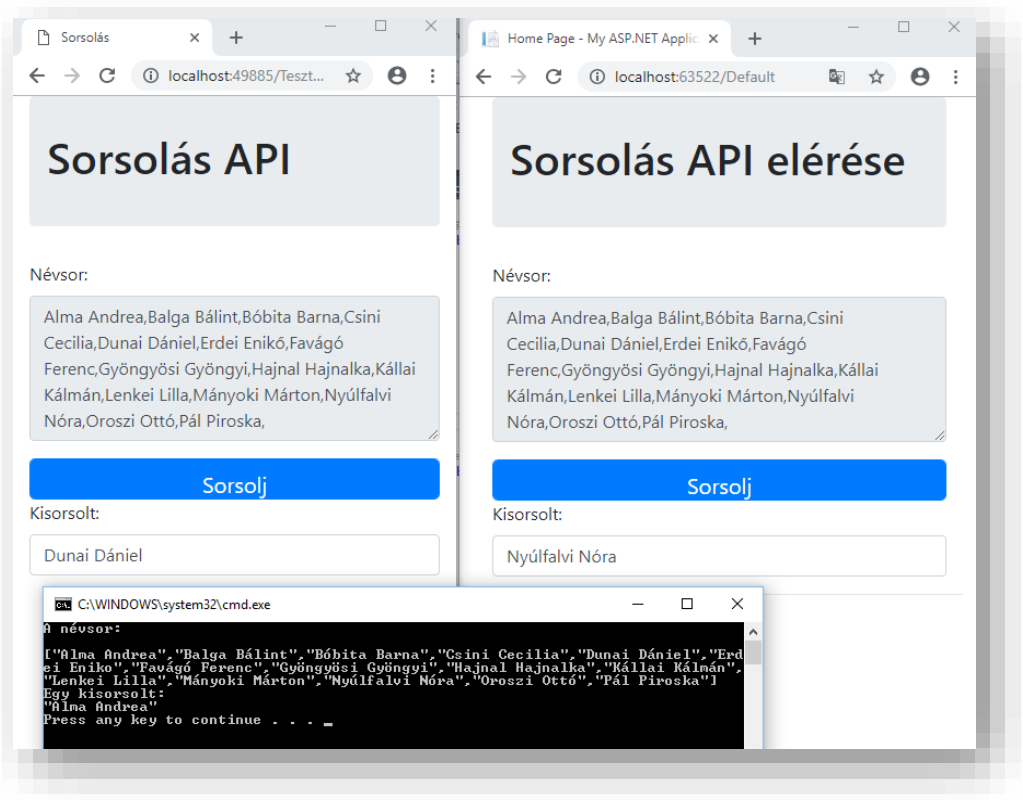
```
static void Main(string[] args)
        {
            Sorsolas objsync = new Sorsolas();
            Console.WriteLine("A névsor:");
            objsync.GetAll();
            Console.WriteLine("Egy kisorsolt:");
            objsync.GetKiválasztott(1);
        }
```

Megjegyzés:

Futtassuk az alkalmazásunkat, de ne feledkezzünk meg arról, hogy az API-nak is futnia kell ugyanakkor! Az eredmény a feladat kitűzésénél látható!

Letölthető	
SorsolasConsole	https://gitlab.com/csharpttk/gyakorlat/sorsolasconsole

Mindhárom alkalmazásunkat egyidőben is futtathatjuk, ahogy az a következő ábrán látszik!



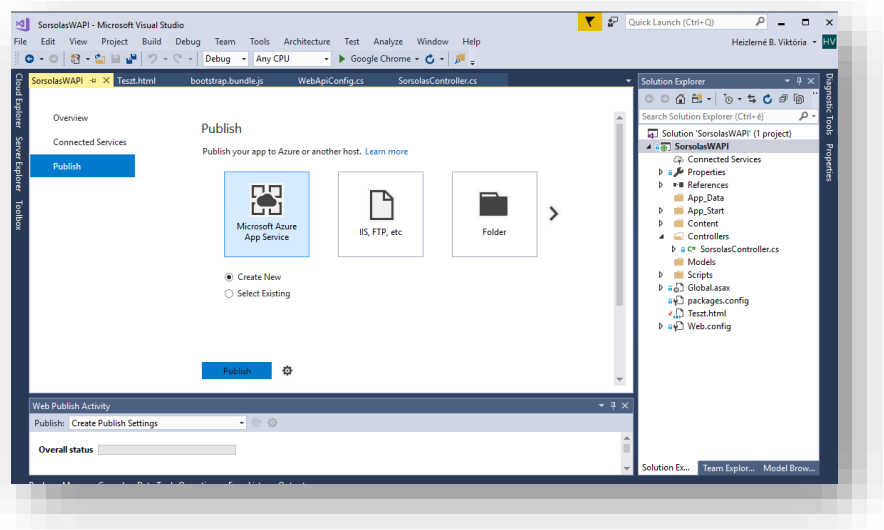
4. Feladat:

A Web API alkalmazásunkat publikáljuk az Azurba, így nemcsak a lokális gépen futó alkalmazásaink érhetik el!

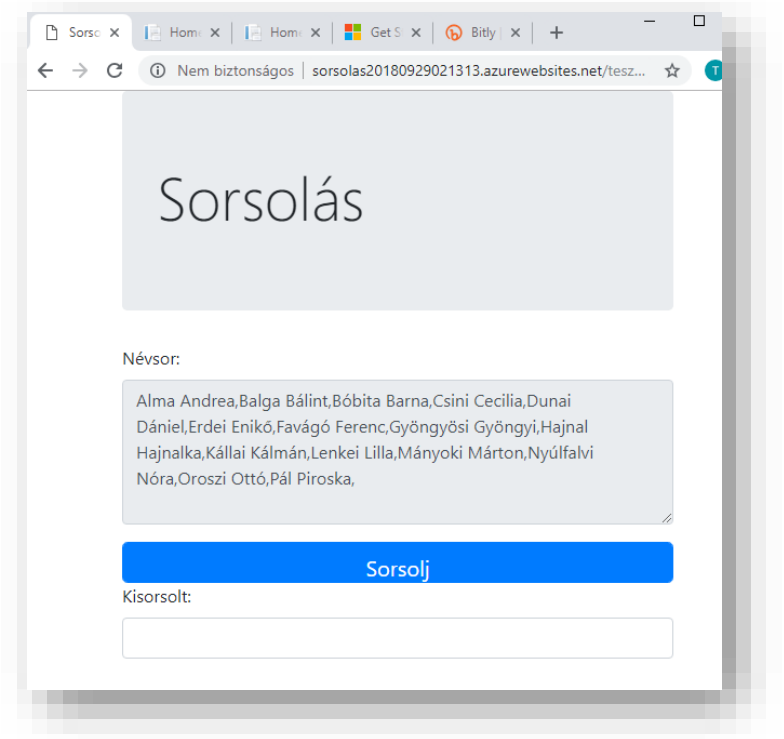
Megjegyzés:

Szükség van egy Azure azonosítóra – lehet free is! https://azure.microsoft.com/en-us/free/

Nyissuk meg a Web Api alkalmazásunkat a Visual Studio-ban, majd a Build menüponton keresztül válasszuk a Publish-t és készítsünk új Azure API-t.



Állítsuk be az Azure hozzáférésünket és kérjünk új resource groupot, plant. (Figyeljünk, hogy a Free változatnál maradjunk!) Ezután automatikusan létrejön a felhőben az API-nk, amit azonnal használni is lehet!



Megjegyzés:

Az Azure szolgáltatást ezen a címen leállítottuk!

Utószó

Természetesen tisztában vagyunk azzal, hogy ebből a gyakorló könyvből nem lehet teljeskörűen elsajátítani a C# programozási nyelvet. Még az első kötetben előforduló elmélet begyakorolásához sem elég egy ekkora tananyag! Igaz, nem is volt ez, nem is lehetett ez a célunk! Mindig az első lépések a legnehezebbek és talán ebben sikerült segítséget adnunk a példa feladatok és kódok bemutatásával! Reméljük kedved kaptak az olvasók a nyelv további lehetőségeinek felfedezéséhez!

Multiplatform lehetőségek, professzionális fejlesztőeszköz, mi kellhet ennél több, hogy rabjai legyünk ennek a területnek?

Sok sikert kívánunk ehhez a munkához! Reméljük tanítványaikat is magukkal ragadják ebbe a varázslatos világba, ahol megnyílik a lehetőség a konzolos alkalmazás írásától, a webes alkalmazásokon, a felhőszolgáltatásokon át a legmodernebb IoT alkalmazások fejlesztéséig!