Eseményvezérelt alkalmazások: 2. gyakorlat

Szavak számlálása

Készítsünk egy konzolos alkalmazást, amely beolvassa egy szöveges fájl tartalmát! Távolítsuk el a szöveg elejéről és végéről a nem betű karaktereket, majd készítsünk statisztikát az így keletkezett szavakról: írjuk ki a konzolra, melyik szó hányszor fordul elő, az előfordulások szerint csökkenő sorrendben.

A Visual Studio 2019 elindítása után válasszuk a `Create a new project` menüpontot.

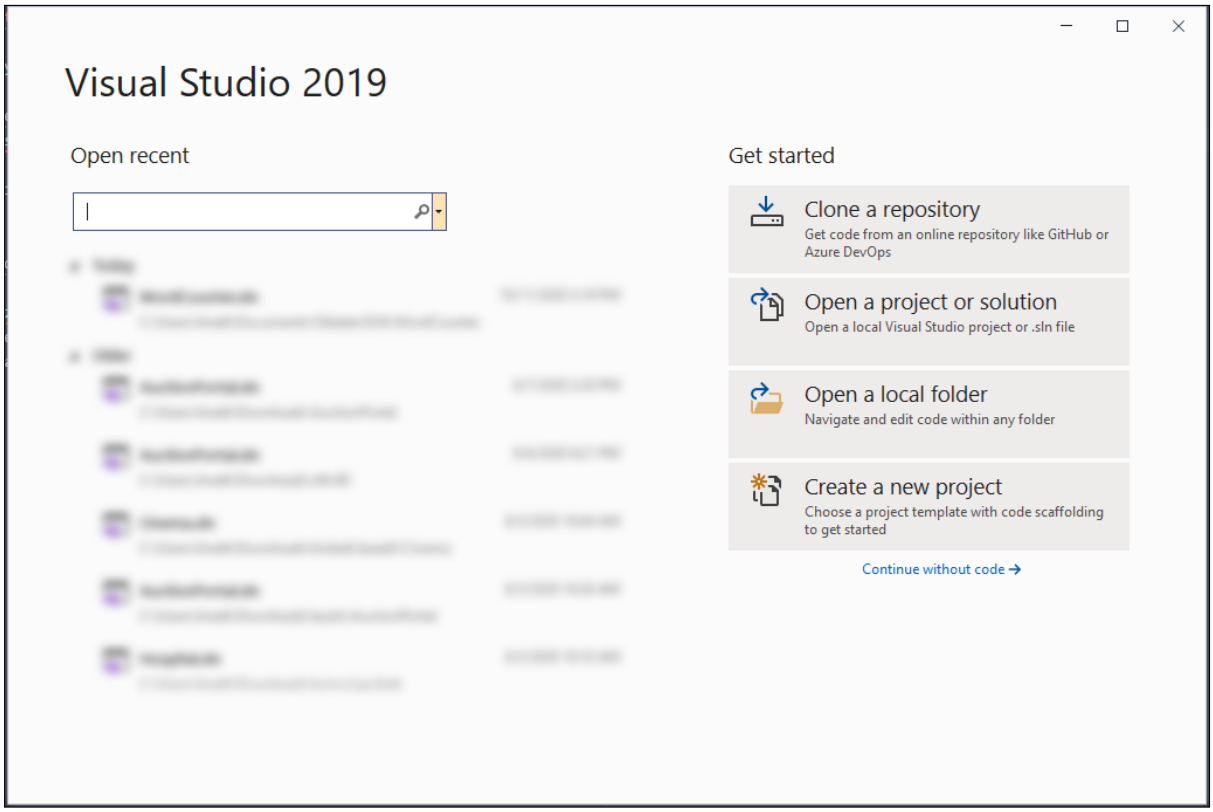

Figure 1: Új solution létrehozása

A következő ablakban válasszuk ki a **Console Application** projektsablont.

> Ne a *Console App (.NET Framework)* sablont válasszuk, ugyanis az .NET 5 / .NET Core helyett a korábbi .NET Frameworkre épülő projektet fog létrehozni.

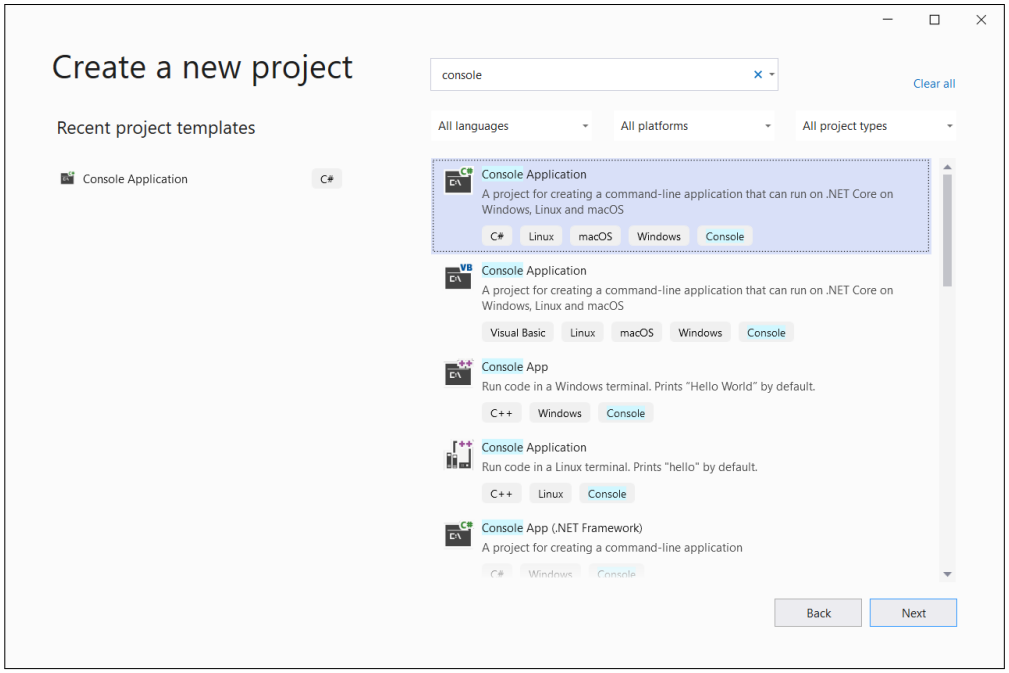

Figure 2: Új projekt létrehozása

A Next gombra való kattintás után megadhatjuk a projekt és a solution nevét. A .NET megoldásokban (*Solution*) gondolkodik, egy Solution több projektet is tartalmazhat. Alapértelmezetten a solution neve megegyezik az első projekt nevével, de ezt lehet módosítani.

Tovább lépve kiválaszthatjuk a cél keretrendszert: válasszuk a hosszú távú támogatással (LTS) rendelkező .NET Core 3.1 verziót, de az újabb .NET 5 is megfelelő.

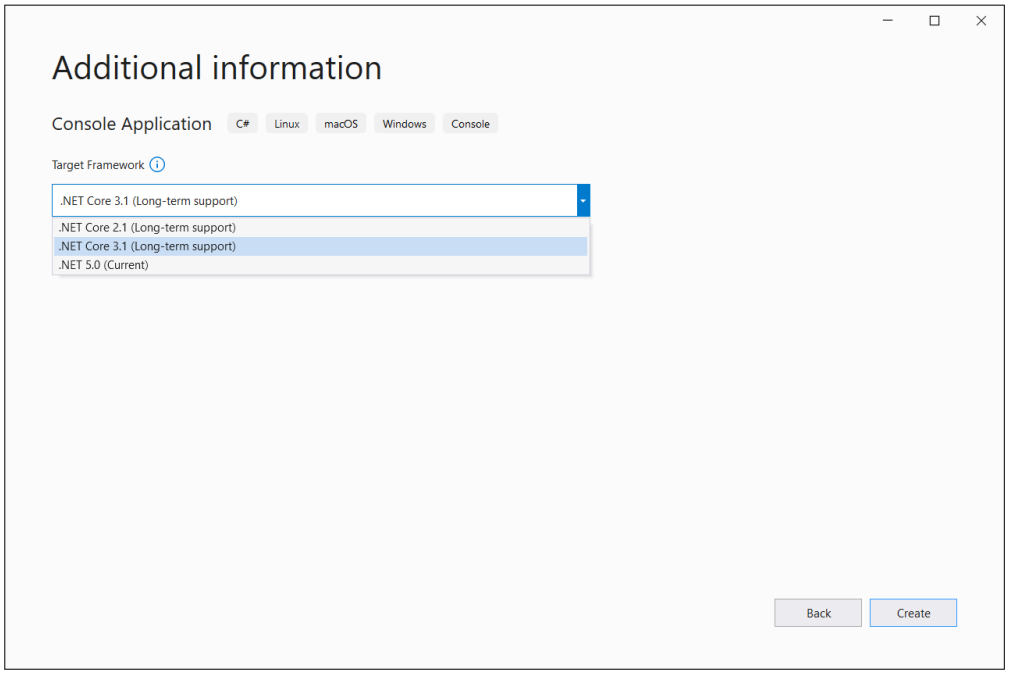

Figure 3: Cél keretrendszer kiválasztása

Az induló projekt a *Solution Explorer*ben a projekt nevén jobb klikk, majd a `Set as StartUp Project` menüpontot választva adható meg. A sablon mindössze egy `Program` nevű osztályt generál, amelyben a statikus `Main` függvény található.

A fájlbeolvasáshoz és a számítások elvégzéséhez hozzunk létre egy új fájlt `Statistics.cs` néven. Ebben automatikusan létrejön egy osztály ugyanezen a néven. Készítsünk egy `FileContent` nevű tulajdonságot (*property*), amely legyen publikus, a típusa legyen `String`, és rendelkezzen egy publikus `get` és egy privát `set` accessorral ("elérő")!

Fájl tartalmának beolvasása

A beolvasás két lépésből áll. Első lépésben bekérünk a felhasználótól egy érvényes abszolút útvonalat egy szöveges (txt kiterjesztésű) fájlhoz. Ezt végezzük a `Main` függvényben, amely addig olvas be a konzolról, amíg egy létező szöveges fájlt nem kapunk. A beolvasást a `Console.Readline()` függvénnyel végezzük. A kiterjesztést a `System.IO.Path.GetExtension()` metódussal, a létezés ellenőrzését a `System.IO.File.Exists()` metódussal végezzük.

A második lépés a fájl tartalmának beolvasása. Ehhez készítsük el a `Load` nevű függvényt a `Statistics` osztályban, amely paraméterül a beolvasott útvonalat várja. A metódusban a `FileContent` értékét írjuk felül a `System.IO.File.ReadAllText(path)` hívással, amely stringként adja vissza a fájl tartalmát. A `ReadAllText()` `System.IO.IOException` kivételt okozhat, ezt azonban majd csak a hívó metódusok fogják elkapni.

Hívjuk meg a `Main`-ben a `Load` függvényt, és egy `try-catch` blokkban kezeljük a kivételt! A `catch` ágban írjuk ki a kivétel üzenetét (`Message`) a `Console.WriteLine()` paranccsal, majd lépjünk ki egy nullától különböző értékkel, így jelezve, hogy valami nem megfelelően zajlott a program futása közben. Például mínusz egyes hibakód esetén: `return -1;`. Ehhez a `Main` függvény visszatérési típusát változtassuk `int`-re és a helyes működést reprezentáló nullás értékkel térjünk vissza a függvény végén.

Szavak előfordulásának megszámlálása

A szövegen végzett műveletek elvégzéséhez definiáljunk egy `CountWords` nevű metódust a `Statistics` osztályban! A szavak előfordulásainak számát egy `String - int` párokat tartalmazó konténerben fogjuk tárolni (`Dictionary<String, int>`). Ezt a konténert is tulajdonságként vegyük fel, hasonlóan a `FileContent`-hez. Adjuk neki a `WordCount` nevet, és inicializáljuk az osztály konstruktorában.

A `CountWords`-ben a szöveget tördeljük szavakra a `Split()` függvény hívásával! Ha ezt paraméter nélkül hívjuk meg, automatikusan a *whitespace*-eknél fogja szétvágni a szöveget. A `Split()` eredménye egy string tömb (`words`). A tömbben ekkor még lehetnek üres stringek, ezeket egy lambdakifejezéssel távolítsuk el a tömbből:

```
words = words.Where(s => s.Length > 0).ToArray();
```

A `words` tartalmát tehát felülírjuk a nem üres stringekkel. A `Where` hívás egy `IEnumerable`-t ad vissza, amit tömbbé kell konvertálnunk az értékadáshoz (`ToArray()`).

A maradék szavakon iteráljunk végig. A szavak elejéről vegyük le az összes nem betű karaktert. A karaktereket a `Char.IsLetter()` metódussal vizsgáljuk meg! Az aktuális szóból a `Remove` metódus hívásával távolíthatjunk el karaktereket. Ennek paraméterként adjuk meg az eltávolítás kezdőindexét (ez a 0. indexű elem lesz), és az eltávolítandó karakterek számát. Ne felejtsük el mindig vizsgálni, hogy van-e még karakter a szóban, így kerüljük el a túlindexelést.

```
while (words[i].Length > 0 && !Char.IsLetter(words[i][0]))
{
    words[i] = words[i].Remove(0, 1);
}
```

Ugyanezt tegyük meg fordítva, a szó utolsó karakterétől kezdve. A `^1` indexeléssel hivatkozhatunk a szó utolsó karakterére (természetesen a szó hosszából számított indexelés is jó megoldás.)

```
while (words[i].Length > 0 && !Char.IsLetter(words[i][^1]))
{
    words[i] = words[i].Remove(words[i].Length - 1, 1);
}
```

Alternatív megoldásként a nem betű karakterek eltávolítására használhatjuk a LINQ nyújtotta lehetőségeket. A `words[i].SkipWhile(...)` metódusnak paraméterül megadhatunk egy feltételt, hogy addig hagyjuk el a string karaktereit, amíg ez teljesül. A karaktereket továbbra is a `Char.IsLetter()` metódussal vizsgáljuk meg. Ezt a köztes eredményt a `Reverse()` metódussal megfordítjuk, majd a megfordított tömbön is elvégezzük a korábbi műveletet. Ezután ne felejtsük el visszafordítani az eredményt. Az eredményt a `String.Concat(...)` metódussal tudjuk visszaalakítani egyszerű stringgé.

```
string.Concat(
words[i]
.SkipWhile(c => !char.IsLetter(c))
.Reverse()
.SkipWhile(c => !char.IsLetter(c))
.Reverse()
);
```

Ezután lehetséges, hogy újabb üres string keletkezett (pl. egy évszám esetén). Ezt vizsgáljuk meg a `String.IsNullOrEmpty()` függvénnyel, és ha igazat kapunk, lépjünk tovább a következő szóra!

Az aktuális szót alakítsuk át a `ToLower()` függvénnyel, hogy csak kisbetűket tartalmazzon, így elkerüljük a kis- és nagybetűs különbségekből adódó duplikációkat.

Vizsgáljuk meg, hogy a szó benne van-e már a `WordCount` kulcsai között (`ContainsKey()`). Ha igen, növeljük a szóhoz tartozó értéket 1-gyel (használható az index operátor `[]`), egyébként vegyük fel a szót a `WordCount`-ba az `Add()` metódussal, értéke 1 legyen.

A `Main`-ben ezután hívjuk meg `CountWords` függvényt. A Dictionary egy rendezetlen típus, ezért a kiírás előtt rendeznünk kell az adatokat az előfordulások száma szerint. Rendezzük az értékek szerint `WordCount`-ot az `OrderByDescending()` metódussal, amelynek a rendezés feltételét egy lambdakifejezésként adjuk át, majd az eredmény `IEnumerable`-t mentsük el egy lokális változóba.

A rendezést LINQ lekérdezéssel, többféle szintaxissal is végezhetjük.

1) *Method Syntax:*

```
var pairs = statistics.WordCount.OrderByDescending(p => p.Value);
```

2) *Query Syntax:*

```
var pairs = from pair in WordCount
            orderby pair.Value descending
            select pair;
```

A két módszer ugyanazt az eredményt adja. Ezután a `pairs`-en végigiterálhatunk egy `foreach` ciklussal, hogy kiírjuk a kulcs-érték párokat.

```
foreach (var pair in pairs)
{
    Console.WriteLine($"{pair.Key}: {pair.Value}");
}
```

Eredmény szűrése

Tegyük paraméterezhetővé, hogy csak a felhasználó által megadott minimális előfordulás számot (`int minOccurrence`) és karakterszámot (`minLength`) elérő szavakat jelenítsük meg az eredmények között.

A paramétereket a felhasználótól az alkalmazás kérje be, a konzolról történő olvasáskor ügyeljünk arra, hogy a felhasználó nem csak egész számot adhat meg (`int.Parse()` vagy `int.TryParse()`). A kódredundancia csökkentése érdekében készítsünk egy `static int ReadInt(string message)` segéd eljárást a `Program` osztályba, amely sikeresen beolvas egy egész számot a konzol inputról (újból próbálkozik, ha nem sikerült).

A `pairs` gyűjteményt szűrjük az így kapott minimális előfordulás szám és karakterszám szerint:

```
var pairs = statistics.WordCount
    .Where(p => p.Value >= minOccurrence)
    .Where(p => p.Key.Length >= minLength)
    .OrderByDescending(p => p.Value);
```