

# Fájlok és könyvtárak kezelése

## Összefoglaló

Fájlok és könyvtárak kezelésére számos eszköz érhető el a **System.IO** névtéren keresztül. A projekt beállításától függően ennek használatba vétele vagy automatikusan megtörténik, vagy manuálisan, a fájl elején meg kell tennünk.

```
using System.IO;
```

## Könyvtárak kezelése

A könyvtárak kezeléséhez szükséges műveleteket a **Directory** osztályon keresztül érjük el. Könyvtárat hozhatunk létre a **CreateDirectory** és törölhetünk a **Delete** metódusokkal. Utóbbinál második paraméterként **true** értéket adva annak teljes tartalmát törölni tudjuk, minden benne lévő könyvtárral és fájllal együtt. Egy könyvtár létezését ellenőrizhetjük az **Exists** metódussal. Egy adott elérési útvonalon lévő könyvtár összes alkönyvtárát a **GetDirectories** metódussal, a benne lévő összes fájlt a **GetFiles** metódussal kérhetjük le, utóbbinál második paraméterként egyszerű szűrőfeltétel is megadható a **\*** vagy a **?** karakter használatával (pl. **"\*.txt"**, ami minden **.txt** kiterjesztésű fájlt visszaad). A metódusok első paramétereként megadott útvonal minden esetben egy **string**, amelyben kikapcsolhatjuk a speciális karakterek értelmezését a karakterlánc előtt elhelyezett **@** jellel (ellenkező esetben az elérési útvonalban gyakran szereplő **\** karakter hibát okozna, így azokat minden előfordulás esetén escapelni kellene).

```
using System.IO;
namespace FilesAndDirectories
{
    class Program
    {
        static void Main(string[] args)
        {
            Directory.CreateDirectory(@"C:\MyFolder\MyFirstFolder");
            Directory.CreateDirectory("C:\\MyFolder\\MySecondFolder");
            string[] allFolders = Directory.GetDirectories(@"C:\MyFolder");
            // { "C:\\MyFolder\\MyFirstFolder", "C:\\MyFolder\\MySecondFolder" }
            string folderToDelete = "C:\\MyFolder";
            if (Directory.Exists(folderToDelete))
            {
                Directory.Delete(folderToDelete, true);
            }
        }
    }
}
```

## Fájlok kezelése

Fájlok kezeléséről kétféle értelemben beszélhetünk: *fájlrendszer szintű kezelés* alatt értjük a fájlok létrehozását, másolását, törlését, adataik lekérdezését, míg *fájlkezelés* alatt egy konkrét fájl tartalmának feldolgozását vagy módosítását.

A **File** osztály metódusai mind fájlrendszer szintű kezelést, mind fájlkezelést lehetővé tesznek. A **Copy** metódus az első paraméterként megadott forrásfájlt másolja a második paraméterként megadott célfájlra. A **Move** működése hasonló, de ez áthelyezi a forrásfájlt. A **Delete** a paraméterben megadott elérési úton lévő fájlt törli. Egy adott útvonalon feltételezett fájl létezését az **Exists** metódussal tudjuk ellenőrizni.

```
string original = @"C:\SourceFolder\mySource.txt";
string copy = @"C:\TargetFolder\myTarget.txt";
File.Copy(original, copy);
if (File.Exists(original))
{
    File.Delete(original);
}
File.Move(copy, original);
```

Egy szöveges fájl sorait, valamint teljes tartalmát beolvashatjuk a **ReadAllLines** és a **ReadAllText** metódusokkal. Szöveges fájlhoz fűzhetünk további sorokat vagy tartalmat az **AppendAllLines** és az **AppendAllText** metódusokkal. Amennyiben új fájlba szeretnénk sorokat vagy tartalmat írni (vagy felül kívánjuk írni a már benne lévő tartalmat), használhatjuk a **WriteAllLines** és a **WriteAllText** metódusokat. Minden esetben megadható paraméterként a felhasznált kódolás is, ami .NET keretrendszer használata esetén az UTF-8 (egyéb esetben a **System.Text.Encoding** valamely értéke használható).

```
string fileName = "heisenberg.txt";
string text = "You clearly 'dont know who 'youre talking to, so let me clue you in.\nI am not in danger, Skyler.\nI am the danger.";
string[] lines = { "A guy opens his door and gets shot, and you think that of me?", "No!", "I am the one who knocks!" };
File.WriteAllText(fileName, text, System.Text.Encoding.ASCII);
File.AppendAllLines(fileName, lines);
string quoteAsText = File.ReadAllText(fileName);
// "You clearly 'dont know who 'youre talking to, so let me clue you in.\n...\nI am the one who knocks!\n"
string[] quoteFromArray = File.ReadAllLines(fileName);
// { "You clearly 'dont know who 'youre talking to, so let me clue you in.", ... "I am the one who knocks!" }
```

## Fájlok mint folyamatok

Lehetőségünk van fájlokat folyamként (stream) kezelni. Ez azt jelenti, hogy egy szöveges fájl tartalmát képzelhetjük karakterek „időben változó sorozatának”, amelynek feldolgozása hatékonyabb lehet például memória-foglalás szempontjából, hiszen nem feltétlenül szükséges egy fájl teljes tartalmát egyszerre a memóriában tartanunk, hanem feldolgozhatjuk azt részletenként is. Megjegyezzük, hogy a **File** osztály egyébként szintén az alábbi két osztályt használja a fájlok feldolgozása során.

Szöveges fájlok folyamként írását teszi lehetővé a [StreamWriter](#) osztály. Az íráshoz használt objektum létrehozásakor adjuk meg a célfájl elérési útvonalát. Opcionálisan megadható még, hogy már létező fájl esetén hozzáfűzés történjen-e (`true`), továbbá beállítható a kódolás is a fentebb bemutatott módon. A [Write](#) és [WriteLine](#) metódusok a konzolnál megismertekhez hasonlóan működnek (vagyis a [WriteLine](#) sortörés karakterrel zárja a paraméterként kapott szöveget). A folyamat végén ne felejtsük el bezárni a folyamatot a [Close](#) metódussal, amely leüríti a folyam által használt puffert (ennek hiányában elképzelhető, hogy a kiírt adatok egy része nem jelenik meg a fájlban, illetve nyitott fájlhoz más folyamat csak korlátozottan férhet hozzá)!

```
StreamWriter myWriter = new StreamWriter("elcamino.txt", true);
myWriter.WriteLine("- What kind of pizza do you like, Jesse?");
myWriter.WriteLine("- Pepperoni.");
myWriter.Write("- Pepperoni, sure, classic. ");
myWriter.WriteLine("I like that too.");
myWriter.Close();
```

Egy szöveges fájl folyamként olvashatunk a [StreamReader](#) osztállyal. Az objektum létrehozásakor adhatjuk meg a fájl elérési útvonalát, valamint opcionálisan a kódolását. Az [EndOfStream](#) tulajdonság megadja, hogy van-e még olvasható adat a folyamban (másképp fogalmazva, hogy a fájl végére értünk-e az olvasással). Az alábbi kódrészlet a megnyitott fájlból addig olvas egy-egy sort a [ReadLine](#) metódussal, amíg a folyamat képező fájl végére nem ér.

```
StreamReader myReader = new StreamReader("elcamino.txt", System.Text.Encoding.ASCII);
while (!myReader.EndOfStream)
{
    string oneLine = myReader.ReadLine();
    Console.WriteLine(oneLine);
}
myReader.Close();
```

## Bináris fájlok feldolgozás folyamként

Nem minden adatot célszerű szöveges formában tárolnunk, mivel ilyenkor a tárigény sokszorosa lehet a valóban szükségesnek. Például az 1234567890 egész szám szöveges formában tárolásához 10 karakter (összesen 20 bájt) szükséges, miközben az érték gond nélkül ábrázolható egy 32 bites (4 bájtos) egészként is. A [BinaryReader](#) és [BinaryWriter](#) osztályok segítségével bármilyen C# alaptípust (primitívet) bináris formában olvashatunk vagy írhatunk. A két osztály használata némileg hasonló a már bemutatott [StreamReader](#) és [StreamWriter](#) osztályokéhoz, ezért a konkrét metódusok részletes ismertetése helyett csak egy példakódot mutatunk néhány alaptípusú érték fájlba írásáról és visszaolvasásáról. Eltérés az előbbiekhöz képest, hogy nem tudjuk közvetlenül a [BinaryReader](#) és [BinaryWriter](#) példányosításakor megadni a fájl elérési útvonalát, hanem előbb egy [FileStream](#) objektumot kell létrehoznunk, amely a bináris fájl reprezentálja. Fontos, hogy beolvasáskor helyes sorrendben értelmezzük a fájlban lévő adatokat, ellenkező esetben az eredetitől eltérő értéket kaphatunk eredményül bármilyen hibaüzenet nélkül! A példa egy ilyen esetet mutat be.

```
byte oneByte = 1;
bool oneBoolean = false;
char oneChar = 'A';
double oneDouble = 1.0;

FileStream writerStream = new FileStream("somedata.bin", FileMode.Create);
BinaryWriter myWriter = new BinaryWriter(writerStream);
myWriter.Write(oneByte);
myWriter.Write(oneBoolean);
myWriter.Write(oneChar);
myWriter.Write(oneDouble);
myWriter.Close();
writerStream.Close();

FileStream readerStream = new FileStream("somedata.bin", FileMode.Open);
BinaryReader myReader = new BinaryReader(readerStream);
bool oneBooleanRead = myReader.ReadBoolean();           // true
double oneDoubleRead = myReader.ReadDouble();           // 8.2212523467983425E-320
byte oneByteRead = myReader.ReadByte();                  // 240
char oneCharRead = myReader.ReadChar();                  // '?'
myReader.Close();
readerStream.Close();
```

## Feladatok

1 Írjunk programot, amely feldolgozza és megfelelő színekkel megjeleníti egy szöveges fájl tartalmát. A fájl sorai az alábbi példához hasonló formátumúak, ahol az adott sor színe a # jel előtt, a sor szövege a jel után található.

### Példa

```
Red#Lorem ipsum dolor sit amet, consectetur adipiscing elit.  
Blue#Suspendisse eros erat, ornare quis magna vitae, rutrum interdum turpis.  
Green#Morbi pellentesque ut sem sed dapibus.
```

Megjegyzés: A példához hasonló *dummy* szöveget generálhatunk tetszőleges hosszúságban ezen az oldalon: [www.lipsum.com](http://www.lipsum.com).

2 Készítsünk lottóhúzást szimuláló és naplózó alkalmazást. Egy lottóhúzás során 90 számból pontosan ötöt húznak ki véletlenszerűen (ismétlések nélkül). Készítsük el az adott heti húzást szimuláló algoritmust, majd mentjük a nyerőszámokat a mai dátummal együtt egy szöveges fájlba, illetve jelenítsük meg a képernyőn is. Kérdezzük meg a felhasználótól, szeretne-e egy újabb heti húzást szimulálni, és pozitív válasz esetén ismételjük meg a fentieket, de a dátum egy héttel későbbi időpontot mutasson. A szimuláció a felhasználó negatív válaszkor érjen véget.

### Példa a képernyő tartalmára

```
On 2023. 10. 08. numbers were: 74 70 79 43 65  
Another week? [y/n] y  
On 2023. 10. 15. numbers were: 82 81 88 37 47  
Another week? [y/n] n
```

3 Készítsünk alkalmazást egy virtuális hangya útvonalának nyomon követésére. Egy szöveges fájl első sorában a hangya kezdeti pozíciója ( $x$  és  $y$ ), valamint iránya ( $0^\circ$ ,  $90^\circ$ ,  $180^\circ$  vagy  $360^\circ$ ) található szóközökkel elválasztva. Az ezt követő sorokban a go, left vagy right utasítások valamelyikét követően egy egész számot találunk. A go  $k$  utasítás a hangyát  $k$  lépéssel helyezi át a jelenlegi irányától függően egy új koordinátára, a left  $d$  és right  $d$  utasítások pedig a hangya aktuális irányát módosítják  $d$  fokkal, az óramutató járásával egyező vagy fordított irányba.

### Példa a bemeneti fájlra

```
25 10 90  
go 2  
left 90  
go 3  
go 1
```

4 Készítsünk elemző programot az NHANES (National Health and Nutrition Examination Survey) adatbázisból származó adatok feldolgozására, és különböző lekérdezések elkészítésére. Használjuk az NHANES\_1999–2018.csv fájlt, amely az 1999-től 2018-ig tartó időszakban végzett felmérések adatait tartalmazza vesszővel tagolt táblázatos formában. A tábla attribútumai:

- SEQN: az alany egyedi azonosítója (egész)

- SURVEY: a felmérés időszaka (szöveg)
- RIAGENDR: az alany neme (szám, 1=férfi, 2=nő)
- RIDAGEYR: az alany életkora években (szám)
- BMXBMI: az alany testtömegindexe (szám)
- LBDGLUSI: az alany vércukorszintje (szám)

Töltsük be az adatokat egy-egy tömbbe, egy adott alany értékei minden tömbben ugyanarra az indexre kerüljenek.

Válaszoljuk meg az alábbi kérdéseket:

1. Egy adott felmérésben mennyi volt a nők és a férfiak átlagos testtömegindexe?
2. Egy adott felmérésben az alanyok hány százalékának volt 5.6-nál magasabb a vércukorszintje?
3. Egy maximális BMI-vel rendelkező alanynak mennyi a vércukorszintje?
4. A teljes adathalmazban mi a túlsúlyos (legalább 30.0-as BMI) személyek átlagos életkora?