Fájlkezelés

Fájlnak nevezünk minden háttértárolón található adatot, adathalmazt (pl. szövegszerkesztőben megírt dokumentum, stb.). A fájlok: azonos típusú komponensekből felépülő adatszerkezetek.

- a programba az adatokat nem csak billentyűzet vagy egér segítségével lehet bevinni, hanem valamilyen háttértárolón tárolt fájlból is be lehet olvasni, főleg nagy mennyiségű adat esetén
- hasonlóan, a program által előállított adatokat nem csak a képernyőre, hanem fájlba is ki lehet írni
- C#-ban a fájlok közös műveleteit stream-ekben valósították meg: a stream adatfolyam, a memória és egy külső egység közötti adatáramlás véghezvitelére
- mielőtt egy fájllal bármilyen műveletet végeznénk, meg kell azt nyitnunk. A fájl megnyitását követően különféle műveleteket végezhetünk. Ezek közül a legfontosabbak az olvasás és az írás. Olvasásnál a fájlból adatokat olvasunk be a memóriába, ami a gyakorlatban egy előzőleg deklarált változó feltöltését jelenti. Ezzel szemben írásnál a memóriából viszünk ki adatokat a fájlba. Az írást és olvasást összefoglalóan I/O (Input-Output) műveleteknek nevezzük
- a .NET számos osztályt biztosít a fájlkezelésre
- a fájl input-output szolgáltatásokat a **system.IO** névtér osztályai nyújtják

Állományfajták

- Szöveges fájlok – soros hozzáféréssel kezelhetjük
- Bináris fájlok – közvetlen hozzáféréssel kezelhetők: tetszőleges elemükre pozícionálhatunk, akkor is, ha ez az elem valahol a fájl belsejében helyezkedik el
- Xml – XmlTextReader és XmlTextWriter osztályok állnak a rendelkezésünkre (Az XML általános célú leírónyelv, amelynek elsődleges célja strukturált szöveg és információ megosztása az interneten keresztül.)

Szöveges fájlok kezelése

- a szöveges fájlokban levő adatok egyszerű szerkesztőprogramokkal (notepad) megtekinthetők
- a szöveges állományokban az adatok szöveg formátumban vannak
- a szöveges formájú adatok sorokra vannak tördelve
- a szöveges fájl input-output műveletek osztályai a **StreamReader** és **StreamWriter** osztályok
- ehhez szükségünk van a **System.Text** névtérre

Szöveges fájl megnyitása olvasásra

- text fájlok olvasásra való megnyitására a **StreamReader** osztályból kell példányosítani
- a StreamReader osztály konstruktorának paraméterként meg kell adni a megnyitandó fájl nevét (kötelező) és meg lehet adni a kódlapot, amellyel a fájl tartalma íródni fog (alapértelmezésként a Default, amit a Windows aktuálisan is használ)
- ha a fájl nem a program saját könyvtárában van, akkor a teljes elérési utat is meg kell adni
- az elérési útvonalnál vigyázni kell, mert a backslash (‘\’) karakternek speciális jelentése van, ezért egyetlen backslash leírásához dupla ‘\\’ kell, vagy használjuk a @ jelet az elérési út előtt, amivel az utána következő sztring literál minden karakterét normális karakterként fogja értelmezni
  “C:\\Mappa1\\Mappa2\\proba.txt” vagy @”C:\Mappa1\Mappa2\proba.txt”

Pl.
```
StreamReader f = new StreamReader("c:\\proba.txt", Encoding.Default);
```

- előfordulhat, hogy a fájl tartalmának kiírásakor az ékezetes karakterek helyett kérdőjel jelenik meg. Ez azért van, mert az éppen aktuális karaktertábla nem tartalmazza ezeket a karaktereket, ez tipikusan nem magyar nyelvű operációs rendszer esetén fordul elő: megadjuk a kódlapot:

Pl.
```
StreamReader f = new StreamReader (@"c:\proba.txt",Encoding.GetEncoding("iso-8859-2"));
```

- szöveges fájl olvasásra való megnyitására használhatjuk a File osztály OpenText metódusát is

Pl.
```
StreamReader f = File.OpenText(@"D:\GME\tanszek-2013-2014\C#\pr1.txt");
```

Szöveges fájl olvasása

- miután megnyitottuk a fájlt olvasásra, utána olvashatjuk a tartalmát
- a textfájlokban az adatok sorokra vannak bontva
- egy lépésben általában egy sort szoktunk kiolvasni a **ReadLine()** metódussal
  Pl.
  ```
  StreamReader f = new StreamReader("c:\\proba.txt");
     string s = f.ReadLine();
  ```
- minden ReadLine()-al történő olvasási művelet során automatikusan lépünk a fájlban a következő sorra
- a ReadLine() többszöri alkalmazása révén eljutunk a fájl végéig
- karakterenként is kiolvashatjuk a fájl tartalmát a Read() függvénnyel, melynek visszatérési értéke a kiolvasott karakter kódja
- a Read()-el történő olvasáskor automatikusan lépünk a fájlban a következő karakterre
  Pl.
  ```
  char c = (char)f.Read();
  ```

Fájl végének elérése

- a fájlból való folyamatos olvasás esetén előbb-utóbb elérjük a fájl végét
- a fájl végének elérésekor a kiolvasott sztring null értéket vesz fel

Pl.
```
string s= f.ReadLine();
while(s!=null)
  {   Console.WriteLine(s);
      s=f.ReadLine();
  }
```

- fájl végének elérését le tudjuk kérdezni a **Peek()** metódus segítségével: a Peek() a soron következő bájt értékét adja meg anélkül, hogy a pozíciót léptetné; ha nincs következő bájt, vagyis elértük a fájl végét, akkor a Peek() által visszaadott érték **-1**

Pl.
```
string s;
while (f.Peek() != -1)
 {
   s = f.ReadLine();
   Console.WriteLine(s);
 }
```

Pozícionálni nem lehet a szöveges fájlban.

Szöveges fájl megnyitása írásra

- szöveges állományok írásra való megnyitására a **StreamWriter** osztályból kell példányosítani
- a StreamWriter osztály konstruktorának első paraméterként meg kell adni a fájl nevét az elérési útvonallal együtt, második paraméterként megadhatjuk a hozzáfűzés módját, ami lehet: false (ez az alapértelmezett) – létező fájl esetén törli a tartalmát, true – hozzáírásra nyitja meg a fájlt; harmadik paraméterként megadhatjuk a kódlapot, ugyanúgy, mint az olvasásnál

Pl.
```
StreamWriter f = new StreamWriter(@"C:\proba2.txt", true, Encoding.UTF8);
```

  - első paraméter: proba2.txt a fájl neve, amelybe írni szeretnénk
  - második paraméter: true – ha létezik a fájl, akkor annak tartalma megmarad és amit most írunk bele, az annak a végére fog íródni (append)
  - harmadik paraméter: Encoding.Default – a fájlbaírás kódlapját adja meg, ami jelen esetben a Windows-unkban használt kódlap
- a File osztály AppendText és CreateText metódusai is rendelkezésünkre állnak szöveges fájl létrehozására illetve létező fájl megnyitására

Pl. `StreamWriter f = File.AppendText(@"C:\proba.txt");` - megnyitja a fájlt írásra: ha létezett, az állomány végére pozicionál és íráskor oda ír, ha nem létezett, akkor létrehozza

Szöveges fájl írása

- miután megnyitottuk a fájlt írásra, a **Write**() és **WriteLine**() metódusok segítségével írhatunk bele
- a két metódus használata megegyik a konzolos metódusokkal, csak most nem a konzolra, hanem az „f”-el azonosított fájlba írunk

Pl.
```
StreamWriter f = new StreamWriter(@"C:\proba2.txt", true, Encoding.UTF8);
f.WriteLine(Console.ReadLine());
int t = 12;
double u=6.23;
f.Write(t+" "+u);
```

- a WriteLine() a kiírás végén egy sorvége jelet is ír a fájlba, több WriteLine() használata esetén minden kiírás adatai új sorba kerülnek

File bezárása

- megnyitott fájlt a **Close**() metódussal tudunk bezárni
- amennyiben elfelejtünk bezárni egy írásra megnyitott fájlt, akkor bizonyos mennyiségű adatmódosítás elveszhet

Pl. `f.Close();`

Feladatok

1. Egy szöveges állomány minden második sorát írassuk ki a képernyőre.

```
string fnev;
Console.Write("Allomany neve utvonallal egyutt:");
fnev = Console.ReadLine();
StreamReader f = File.OpenText(@fnev);
while (f.Peek() != -1)
{
    Console.WriteLine(f.ReadLine());
    f.ReadLine();
}
Console.ReadKey();
```

2. Töltsünk fel egy szöveges állományt egy csoport névsorral, és minden egyes név után az egyes személyek életkora legyen található. A szöveges állományban található személyek átlag életkorát számold ki.

```
struct szemely
```

```
{
    public string nev;
    public int eletkor;
}


static void Main(string[] args)
{
    szemely a = new szemely();
    StreamWriter f = new StreamWriter(@"feladat2.txt");
    int n;
    Console.Write("Szemelyek szama:");
    n = int.Parse(Console.ReadLine());
    for (int i = 1; i <= n; i++)
    {
        Console.Write("{0}. szemely neve:", i);
        a.nev = Console.ReadLine();
        Console.Write("{0}. szemely eletkora:", i);
        a.eletkor = int.Parse(Console.ReadLine());
        f.WriteLine(a.nev+"_"+a.eletkor.ToString());
    }
    f.Close();
    StreamReader g = new StreamReader(@"feladat2.txt");
    string[] s;
    double atl = 0, sz=0;
    while (g.Peek()!=-1)
    {
        s = g.ReadLine().Split('_');
        a.nev= s[0];
        a.eletkor = int.Parse(s[1]);
        atl = atl + a.eletkor;
        sz++;
    }
    g.Close();
            Console.WriteLine("Csoport atlageletkotra: {0}", atl/sz);
    Console.ReadKey();
}
```

Bináris fájlok kezelése

A bináris állományok bináris formátumban tárolják az adatokat ugyanúgy, mint ahogy a memóriában vannak tárolva, ezért az adatok írása a memóriából a bináris fájlokba, illetve fordítva, a bináris fájlokból az adatok visszaolvasása a memóriába nagyon gyors

Bináris fájl megnyitása

- egy fájl megnyitásához példányosítani kell a **FileStream** osztályt
- egy FileStream objektum létrehozásakor a FileStream konstruktorának leggyakrabban 4 paramétert szoktak megadni:
    1. fájl neve (ha szükséges az elérési útvonallal együtt) - kötelező
    2. megnyitás módja (FileMode)

3. fájlelérés (FileAcces)
4. megosztás módja (FileShare)

1. Fájl neve:
   – ez az első paraméter
   – ha a fájl nem a program saját könyvtárában van, akkor a teljes elérési utat is meg kell adni
   – az elérési útvonalnál vigyázni kell, mert a backslash (‘\’) karakternek speciális jelentése van, ezért egyetlen backslash leírásához dupla ‘\\’ kell, vagy használjuk a @ jelet az elérési út előtt, amivel az utána következő sztring literál minden karakterét normális karakterként fogja értelmezni
   pl. “C:\\Mappa1\\Mappa2\\proba.txt” vagy @”C:\Mappa1\Mappa2\proba.txt”
2. Megnyitás módja:
   - a FileMode enum-nak a következő értékei lehetnek:
   - Open – létező fájl megnyitása, ha nem létezik, akkor kivételt dob a program
   - Append – létező fájlt megnyit és automatikusan a végére áll, illetve ha nem létezik, akkor létrehozza
   - Create – új fájlt hoz létre, ha már létezett, akkor törli a tartalmát
   - OpenOrCreate – megnyit egy létező fájlt, ha nem létezik, akkor létrehozza
   - Truncate – megnyit egy létező fájlt és törli a tartalmát
3. Fájlelérés:
   - beállíthatjuk, hogy pontosan mit akarunk csinálni az állománnyal
   - Read – olvasunk a fájlból
   - Write – írunk a fájlba
   - ReadWrite – olvasásra és írásra nyitjuk meg a fájlt
4. Megosztás módja:
   - beállíthatjuk, ahogy más folyamatok férnek hozzá a fájlhoz:
   - None – amíg a fájl meg van nyitva, addig más folyamat nem férhet hozzá a fájlhoz
   - Read – más folyamat csak olvashatja a fájlt
   - Write – más folyamat csak írhatja
   - ReadWrite – más folyamat írhatja is és olvashatja is a fájlt
   - Delete – nem magát a fájlt, hanem a fájlból törölhet más folyamat

Pl.1.
```
FileStream f = new FileStream(@"c:\proba1.dat", FileMode.Open);
```
Pl.2
```
FileStream f = new FileStream(@"c:\teszt1.dat", FileMode.Open, FileAccess.Read);
```
Pl.3
```
FileStream f = new FileStream(@"c:\teszt2.dat", FileMode.Create,FileAccess.Write);
```

– bináris fájlok megnyitására használhatjuk a File osztály Open(), Create() metódusait is

Olvasás bináris fájlból

- olvasáshoz a **BinaryReader** osztályt használjuk
- egy BinaryReader objektum létrehozásakor meg kell adni a konstruktornak egy FileStream típusú objektumot:

```
FileStream f = new FileStream(@"c:\proba1.dat", FileMode.Open);
BinaryReader b = new BinaryReader(f);
```

- az olvasás a ReadXXXX() függvény segítségével történik (ReadChar, ReadInt32, ReadDouble, stb.)
- megnyitás után az első olvasás a fájl legelső pozíciójától hajtódik végre
- minden olvasási művelettel a következő adatra lépünk a fájlban (például egy ReadInt32() olvasási művelettel 4 bájttal lép előre a fájlban, mert egy Int32 típusú adat 4 bájton van tárolva)

Pl.
```
double d;int a;
a = b.ReadInt32();
d = b.ReadDouble();
```

Fájl végének elérése

- fájl végének elérését le tudjuk kérdezni a **PeekChar**() metódussal
- a PeekChar() metódus a soron következő bájt értékét adja vissza, illetve -1-et, ha elértük a fájl végét
- a PeekChar() a folyambeli aktuális pozíciót nem változtatja meg

Pl. ha a fájlban 20 darab egész szám van:
```
int[] a=new int[20]; int i = 0;
while (b.PeekChar() != -1)
{
    a[i++] = b.ReadInt32();
}
```

Pozicionálás fájlban

- bináris állományoknál lehetőségünk van a fájlmutató állítására: visszatérhetünk egy korábbi adatra, újból kiolvashatjuk, átugorhatunk bizonyos részt feldolgozás nélkül, stb.
- a **Seek**() metódussal tudunk fájlon belül pozícionálni: első paraméterként meg kell adni egy egész számot, második paraméterként pedig a szám értelmezését
- az értelmezést a **SeekOrigin** enum tartalmazza
    - Begin: ha a második paraméter SeekOrigin.Begin, akkor az első szám annak a bájtnak a sorszámát tartalmazza, amelyikre pozícionálni akarunk. Az első bájt sorszáma 0 és a sorszámozás a fájl elejétől kezdődik

- End: az első szám annak a bájtnak a sorszámát tartalmazza a fájl végétől visszafele számolva, amelyikre pozícionálni akarunk. Az utolsó bájt sorszáma 0 és a sorszámozás visszafele halad
- Current: az első szám határozza meg, hogy hány bájttal kell előre (pozitív érték) vagy visszafele pozícionálni (negatív érték) az aktuális pozíciótól mérve

Pl. Minden második egész szám kiolvasása egy bináris fájlból:

```
FileStream f = new FileStream(@"c:\users\user\desktop\proba2.dat", FileMode.Open);
     BinaryReader b = new BinaryReader(f);
     int k = 0;
     long t = f.Length; // t a fájl mérete bájtban
     int[] a = new int[t/8+1]; // mivel egy adat 4 bájtos, így összesen t/4 adat
                               van, nekünk meg cask a felére van szükségünk
     while (b.PeekChar() != -1)
     {
         a[k++] = b.ReadInt32();
         f.Seek(4, SeekOrigin.Current);
     }
```

Írás bináris fájlba

- íráshoz a **BinaryWriter** osztályt használjuk
- egy BinaryWriter objektum létrehozásakor meg kell adni a konstruktornak egy FileStream típusú objektumot:

Pl.
```
FileStream g = new FileStream(@"C:\proba2.dat",FileMode.Append, FileAccess.Write);
BinaryWriter bw = new BinaryWriter(g);
```

- az írás a **Write**() függvény segítségével történik
- megnyitás után a fájlíró az állomány első pozícióján áll, hacsak nem hozzáfűzésre nyitottuk meg az állományt (Append), amikor a fájlíró az állomány végén áll
- minden írási művelet során lépünk a fájlban a megfelelő mennyiségű bájttal (pl. ha egész típusú adatot írunk a fájlba, akkor 4 bájtot lép előre)

Pl. 20 véletlen egész számot írjunk a proba2.dat végére

```
FileStream g = new FileStream(@"C:\ proba2.dat", FileMode.Append);
BinaryWriter bw = new BinaryWriter(g);
Random r = new Random();
for (int i = 0; i < 20; i++)
     bw.Write(r.Next(100));
bw.Close();
```

File bezárása

- a Close() metódussal – akár a BinaryReader vagy BinaryWriter példányt, akár a FileStream példányt is használhatjuk lezárásra

Pl. `bw.Close();`
vagy `g.Close();`

Feladat

1. Töltsünk fel egy bináris állományt egy csoport névsorral, és minden egyes név után az egyes személyek életkora legyen található. A beolvasott bináris állományt bontsuk fel két szöveges állományra aszerint, hogy az illető személy neve meghaladja-e a 12 karaktert.

```
struct szemely
 {
     public string nev;
     public int eletkor;
 }

 static void Main(string[] args)
 {
     szemely a = new szemely();
FileStream f = new FileStream("feladat1.dat", FileMode.Create, FileAccess.Write);
     BinaryWriter bw = new BinaryWriter(f);
     int n;
     Console.Write("Szemelyek szama:");
     n = int.Parse(Console.ReadLine());
     for (int i = 1; i <= n; i++)
     {
         Console.Write("{0}. szemely neve:", i);
         bw.Write( Console.ReadLine());
         Console.Write("{0}. szemely eletkora:", i);
         bw.Write(int.Parse(Console.ReadLine()));
     }
     bw.Close();
     f.Close();

     f= new FileStream("feladat1.dat", FileMode.Open, FileAccess.Read);
     BinaryReader br = new BinaryReader(f);
     StreamWriter f1 = new StreamWriter("sz1.txt");
     StreamWriter f2 = new StreamWriter("sz2.txt");
     while (br.PeekChar()!=-1)
     {
         a.nev= br.ReadString();
         a.eletkor = br.ReadInt32();
         if (a.nev.Length > 12)
             f2.WriteLine(a.nev + "_" + a.eletkor.ToString());
         else
             f1.WriteLine(a.nev + "_" + a.eletkor.ToString());
     }
     br.Close();
     f1.Close();
     f2.Close();
     f.Close();
     Console.ReadKey();
 }
```