C#

Megjegyzések, kommentek

// egysoros komment
/ ez*
egy több sorból
*álló komment */*

Utasítások lezárása

;

Kiírás

Console.Write("Alma"); a kurzor a sor végén marad
Alma_
Console.WriteLine("Körte"); a kurzor a következő sor elejére kerül
Körte
_
Paraméters kiírás:
Console.WriteLine("szöveg {0} {1}", "érték", 2); a paramétereket a vessző után, vesszővel elválasztva kell felsorolni
szöveg étek 2
Kiírás konkatenációval:
Console.WriteLine("Ma" +32+"fok van");
Ma32fokvan Figyelni kell a szóközre!

Változók

típusos
string, char, int, byte, double, bool, DateTime, TimeSpan, és még néhány egész típus

Beolvasás

Console.ReadLine(); Enter leütésére vár
Console.ReadKey() Egy billentyű leütésére vár

Konvertálás

int a=Convert.ToInt32(Console.ReadLine());
int b=int.Parse(Console.ReadLine());

Értékadás

=

Aritmetikai műveletek

\+ - * / % (maradék)

Logikai műveletek

= =, <, <=, >, >=, !=,
!, &&, ||

A Math osztály használata

Pi	*Math.PI*
felfelé kerekítés	*Math.Ceiling(szám)*
lefelé kerekítés	*Math.Floor(szám)*
kerekítés	*Math.Round(szám)*
hatványozás	*Math.Pow(alap, kitevő)*
gyökvonás	*Math.Sqrt(szám)*
minimum	*Math.Min(szám)*
maximum	*Math.Max(szám)*
abszolútérték	*Math.Abs(szám)*
szögfüggvények	*Math.Sin(rad), Math.Cos, Math.Tan, Math.Asin(valós), Math.Acos, Math.Atan*

Véletlenszám

Random rnd=new Random();

int32 tartományban	*rnd.Next()*
0-99	*rnd.Next(100)*
51-99	*rnd.Next(51, 100)*

[0; 1) intervallumban generál *rnd.NextDouble()*

Sztringek

”szigorúan csak macskakörmök között”
” \’Igen\’, válaszolta”
” \”Igen\”, válaszolta”

\’	’
\”	”
\\	\
\n	New Line, sortörés
\t	Tab
\b	BackSpace

szo=’Programozás’

mit csinál	**utasítás**	**eredmény**
3-as indexű elem	*szo[3]*	*g*
3-4 indexű elem	*szo.Substring(tól, db)* *szo.Substring(3,2)*	*gr*
hossza, karakterek száma	*szo.Length*	*11*
levágja a szóközt	*szo.Trim()* *szo.TrimStart()* *szo.TrimEnd()* *szo.Trim(’a’)*	*elejéről és a végéről* *balról* *jobbról* *levágja az ’a’-t*
kisbetűssé alakít	*szo.ToLower()*	*programozás*
nagybetűssé alakít	*szo.ToUpper()*	*PROGRAMOZÁS*
karaktercsere akár több szomszédos karaktert is lehet	*szo.Replace(mit, mire)* *szo.Replace(”ás”,”ó”)*	*programozó*
tartalmazza-e	*szo.Contain(mit)* *szo.Contains(”am”)* *szo.Contains(”i”)*	*true* *false*
megkeres valamit, kezdőindexet adja, ha nincs akkor -1	*szo.IndexOf(”o”)*	*2* *-1*
utolsó előfordulás	*szo.LastIndexOf(”o”)*	*7*
sztringek összefűzése	*string.Concat(str1, str2)*	
. e az utolsó	*szo.EndsWidth(”.”)*	*false/true*
H-val kezdődik-e	*szo.StartsWith(”H”)*	*false/true*
beszúrás adott pozícióba	*szo.Insert(hova, mit)* *szo.Insert(4,”+”*	*Prog+ramozás*
törlés adott pozíciótól adott pozíciótól, adott számú karaktert	*szo.Remove(3)* *szo.Remove(3,2)*	*Pro* *Proamozás*
feldarabol szóköz mentén feldarabol ’o’ mentén a darabokat tömbbe teszi	*szo.Split()* *szo.Split(’o’)*	*Programozás* *Pr gram zás*

Adatszerkezetek

Tömbök: homogén, statikus
típus[] név=new típus[elemek száma];
string[] gyumi=new string[3];
gyumi[0]=”alma”;
gyumi[1]=”körte”;
gyumi[2]=”szilva”;
string gyumi2={”alma”, ”körte”, ”szilva”};
típus[,] név=new típus[sorok száma, oszlopok száma];
int[,] szamok=new int[3, 5];

mit csinál	utasítás	eredmény
hossza:	*gyumi.Length* *szamok.Length*	*3* *15*
sorok száma:	*szamok.GetLength(0)*	*3*
oszlopok száma:	*szamok.GetLength(1)*	*5*
Tömb rendezése:	*Array.Sort(gyumi)*	
Legkisebb elem:	*nev.Min()*	
Legnagyobb elem:	*nev.Max()*	
Elemek összege:	*nev.Sum()*	
Elemek Átlaga:	*nev.Average()*	

Listák: homogén, dinamikus

List<típus> név=new List<típus>();

mit csinál	utasítás	eredmény
lista elemszáma	*nev.Count())*	
kapacitás: (lefoglalt hely az elemeknek)	*nev.Capacity()*	
felesleges foglalás csökkentése	*nev.TrimExcess()*	
listaelem hozzáadása	*nev.Add("narancs")*	*1, alma, citrom, narancs*
elem beszúrása adott pozícióba	*nev.Insert(index, érték)* *nev.Insert(1, "eper")*	*1, eper, alma, citrom, narancs*
elem törlése	*nev.Remove(mit)*	
az adott indexű elem eltávolítása	*nev.RemoveAt(index)*	
lista törlése, üres listát ad vissza	*nev.Clear()*	
lista rendezése	*nev.Sort()*	
lista elemei fordított sorrendben	*nev.Reverse()*	
adott szerepel a listában az adott elem	*nec.Contains(elem)*	*true/false*
azonos típusú kollekció hozzáadása a listához	*nev.AddRange(kollekció)*	
azonos típusú kollekció hozzáadása a listához adott indexű helyre	*nev.InsertRange(index, kollekció)*	
lista tömbbé alakítása	*nev.Array()*	
Legkisebb elem:	*nev.Min()*	
Legnagyobb elem:	*nev.Max()*	
Elemek összege:	*nev.Sum()*	
Elemek Átlaga:	*nev.Average()*	

Vezérlési szerkezetek

Elágazások

```
if(feltétel)
{
        utasítások, ha a feltétel igaz
}

if(feltétel)
{
        utasítások, ha a feltétel igaz
}
else
{
        utasítások, ha a feltétel hamis
}

if(feltétel)
```

```
{
        utasítások, ha a feltétel igaz
}
else if(feltétel)
{
        utasítások, ha a feltétel igaz

}
else
{
        utasítások, ha a feltétel hamis
}

switch(válozónév)
{
case érték1:
  utasítások1
  break;
case érték2:
  utasítások2
  break;
.
.
default:
  utasítások ha egyik érték sem fordult elő, de ez elhagyható
  break;
}
```

Ciklusok

```
for (int i=kezdőérték; i<érték;i=i+lépésköz)
{
        utasítások
}

while (feltétel)
{
        utasítások
}

do
{
        utasítások
}while(feltétel);
```

A while és do-while ciklusoknak, ha van ciklusváltozója, akkor azt a ciklus előtt kell megadni, és a feltételben szerepelnie kell, az utasítások között pedig gondoskodnunk kell a változtatásáról

```
foreach( var item in collectin)
{
        utasítások
}
```

leginkább tömbök, listák bejárására használjuk

Függvények, eljárások

A függvényeket és eljárásokat a főprogramon (Main) kívül helyezzük el.

utasítás	meghívása:
Eljárás esetén nincs visszatérési érték, ezért void.	
static void fvnev(paraméter/ek) *{* *utasítások* *}*	*fvnev(érték/ek)*
Függvény esetén a visszatérési érték típusát meg kell adni. A return kulcsszó után adjuk meg a visszatérési érték típusának megfelelő adatot	*Console.Write(fvnev(érték/ek));* *x=fvnev(érték/ek);*
static visszatérésiértéktípusa *fvnev(paraméter/ek)* *{* *utasítások* *return érték/változó/amit* *visszaküldünk;* *}*	

Objektumok

A **struktúrák** olyan adategységek, amelyekben különböző típusú adatokat foglalhatunk egy adatszerkezetbe. A struktúrákat is a főprogramon kívül, a Program osztályon belül kell létrehozni, A *struct* kulcsszóval, ezután következik a neve, majd a kapcsos zárójelek között a magadott mezők *public* láthatósággal

```
struct Nev{
        public típus mezőnév1;
        public típus mezőnév2;
        public típus mezőnév3;
        public void/típus eljarasnev/fvnév(paraméter/ek)
        {
                utasítások
        }
}
```

használata:

Egyetlen példány létrehozása:

```
Nev példánynév=new Nev();
példánynév.mezőnév1=érték1;
```

.

Több példány létrehozása, pl. tömbbe:

```
struct Jatekos
    {
      public string nev, kaszt;
      public int oles;
    }
    static void Main(string[] args)
    {
      string[] beolvas = File.ReadAllLines("forras.txt");
      Jatekos[] jatekosok = new Jatekos[beolvas.Length];
      string[] darabok = { };
      for (int i = 0; i < jatekosok.Length; i++)
      {
        darabok = beolvas[i].Split('\t');
        //jatekosok[i] = new Jatekos();
        jatekosok[i].nev = darabok[0];
        jatekosok[i].kaszt = darabok[1];
        jatekosok[i].oles = int.Parse(darabok[2]);

      }
```

Osztály
A *class program*-on kívül a *namespace*-en belül

```
class Nev{
        public típus mezőnév1;
        public típus mezőnév2;
        public típus mezőnév3;
        public Nev(string egysor)
        {
        // ez a konstruktor, szerepe a kezdőértékek beállítása
        darabok = egysor.Split('\t');
        mezonev1 = darabok[0];
        mezonev2 = darabok[1];
        mezonev3 = int.Parse(darabok[2]);
        }
        public void/típus eljarasnev/fvnév(paraméter/ek)
        {
                utasítások
        }
}
```

használata:
Egyetlen példány létrehozása:

```
Nev példánynév=new Nev();
példánynév.mezőnév1=érték1;
...
```

Több példány létrehozása, pl. tömbbe:

```
static void Main(string[] args)
{
        string[] beolvas = File.ReadAllLines("forras.txt");
        Nev[] tömbnév = new Nev[beolvas.Length];
        for (int i = 0; i < tömbnév.Length; i++)
        {
                tömbnév[i]=new Nev(beolvas[i]);
        }
```

Fájlkezelés

A fájlok kezeléséhez szükségünk van a using részbe az IO névtér felvételére.

```
using system.IO;
```

A fájlok helye, ha nem adunk meg elérési útvonalat: *Debug* mappa
Útvonal megadása:

```
@"D:\Saját\Fájlok\szöveg.txt"
```

A \ jel speciális karakter, ezért használjuk a @ módosítót.

Beolvasás

1. **Sok sor, soronként egy adat**

Ebben az esetben nincs különösebb dolgunk, mint beolvasni a fájl tartalmát és a szükséges, akkor konvertálni az elemeket. Így egy tömbben tudjuk tárolni a fájl tatalmát.

FileStream-mel:

```
List<int> cipo = new List<int>();
FileStream fs1 = new FileStream("cipomeret.txt", FileMode.OpenOrCreate);
        StreamReader sr1 = new StreamReader(fs1);
                while (!sr1.EndOfStream)
                {
                        cipo.Add(int.Parse( sr1.ReadLine()));
                }
        sr1.Close();
fs1.Close();
```

FileStream nélkül:

```
string[] beolvas1 = File.ReadAllLines("cipomeret.txt");
```

2. Egy sor, sok adattal:

FileStream-mel:

```
FileStream fs2 = new FileStream("szamok.txt", FileMode.OpenOrCreate);
        StreamReader sr2 = new StreamReader(fs2);
                sor= sr2.ReadLine();
        sr2.Close();
fs2.Close();
string[] darabok=sor.Split(',');
int[] szamok=new int[darabok.Length];
for( i=0; i<szamok.Length; i++)
{
        szamok[i]=int.Parse(darabok[i]);
}
```

FileStream nélkül:

```
string[ ] sor = File.ReadAllLine("szamok.txt");
string[] darabok=sor.Split(',');
int[] szamok=new int[darabok.Length];
for( i=0; i<szamok.Length; i++)
{
        szamok[i]=int.Parse(darabok[i]);
}
```

3. sok sor, soronként több adat

Az előző kettő kombinálva.

FileStream-mel:

```
List<int> sorok = new List<int>();
FileStream fs3 = new FileStream("forras.txt", FileMode.OpenOrCreate);
        StreamReader sr3 = new StreamReader(fs3);
                while (!sr3.EndOfStream)
                {
                        sorok.Add(int.Parse( sr3.ReadLine()));
                }
        sr3.Close();
fs3.Close();
string[] darabok=sorok[0].Split(',');          //oszlopok számának meghatározásához
int sorokszama=sorok.Count, oszlopokszama=darabok.Length;
string[,] adatok=new string[sorokszama, oszlopokszama];    //a típust az adatoknak megfelelően választjuk, ha kell majd a ciklusban konvertálunk
for( i=0; i<adatok.GetLength(0); i++)
{
        darabok=sorok[i].Split(',');
        for( j=0; j<adatok.GetLength(1), j++)
        {
                adatok[i,j]=darabok[j];   //ha kell itt kell konvertálni
        }
}
```

FileStream nélkül:

```
string[] beolvas3 = File.ReadAllLines("forras.txt");
string[] darabok=beolvas3[0].Split(',');     //oszlopok számának meghatározásához
int sorokszama=beolvas3.Length, oszlopokszama=darabok.Length;
string[,] adatok=new string[sorokszama, oszlopokszama];    //a típust az adatoknak megfelelően választjuk, ha kell majd a ciklusban konvertálunk
for( i=0; i<adatok.GetLength(0); i++)
{
        darabok=sorok[i].Split(',');
        for( j=0; j<adatok.GetLength(1), j++)
        {
                adatok[i,j]=darabok[j];   //ha kell itt kell konvertálni
        }
}
```

A fájl egyetlen stringbe történő beolvasása:
string összesszöveg=File.ReadAllText();

Írás:
Fájlok írása Stream-ekkel:
Fájlok írásakor egy írót (sw) kell példányosítanunk a StreamWriter osztályból:
StreamWriter sw = new StreamWriter(fs);
Ezt követően írhatunk a fájlba:
sw.Write("alma");
vagy
sw.WriteLine("alma");
Itt is érvényes a Write és a WriteLine közötti különbség: ha a Write-t használjuk, akkor a következő kiírás ugyanabba a sorba fog kerülni, ha pedig a WriteLine-t akkor a következő kiírás új sorba kerül.
Ha befejeztük az írást, akkor ezt is le kell zárni:
sw.Close();

Pl.: Írjuk ki a *hello.txt* fájlba: HELLO!
FileStream fs = new FileStream("hello.txt", FileMode.OpenOrCreat);
StreamWriter sw = new StreamWriter(fs);
sw.Write("HELLO");
sw.Close();
fs.Close();
A FileStream osztály példányosítását el is hagyhatjuk, ekkor a StreamWriter példányosításakor adjuk meg a fájl elérési útvonalát és nevét:
StreamWriter sw = new StreamWriter("filenév elérési útvonallal");
Itt is érvényes: ha nem adunk meg elérési útvonalat, akkor a Debug mappába kerül a fájl.
Pl.:
StreamWriter sw = new StreamWriter("hello.txt");
sw.Write("HELLO");
sw.Close();

1. **Egyetlen sornyi adat kiírása:**

Pl.:Írjuk ki az első 5 pozitív számot, szóközzel elválasztva egymás mellé!

```
StreamWriter sw = new StreamWriter("szamok.txt");
for (int i = 1; i < 6; i++)
        {
                sw.Write("{0}  ", i);
        }
sw.Close();
```

2. **Több sornyi adat kiírása:**

Pl.:Írjuk ki az első 5 pozitív számot, egymás alá!

```
StreamWriter sw = new StreamWriter("szamok.txt");
for (int i = 1; i < 6; i++)
        {
                sw.WriteLine(i);
        }
sw.Close();
```

3. **Több sor, soronként több adat**

Pl.:Írjunk ki 5 sorban soronként 3 számot, szóközzel elválasztva!

```
StreamWriter sw = new StreamWriter("szamok.txt");
for (int i = 1; i < 6; i++)
        {
                for (int j = i; j < i+3; j++)           // egy sornyi szám kiírása
                {
                        sw.Write("{0}  ", j);
                }
                sw.WriteLine();   //ezzel biztosítjuk, hogy új sorba kerüljön a következő három szám
        }
```

sw.Close();

Fájlok kiírása Stream-ek nélkül:

1. **Egyetlen sornyi adat kiírása:**

File.WriteAllText(”filenév elérési útvonallal”, sztring);
Pl.:
File.WriteAllText(”hello.txt”, ”HELLO”);
vagy:
string szoveg = ”HELLO”;
File.WriteAllText(”hello.txt”, szoveg);

Szöveg típusú tömb elemeinek elválasztó karakterrel egyetlen sornyi szöveggé való átalakítását a következő módon végezhetjük el:
string.Join(”elválasztó karakter”, szöveg típusú tömb);
Pl:
string[] adatok = {”1”, ”2”, ”3”, ”4”};
string kiiras = string.Join(”:”, adatok);
File.WriteAllText(”szamok.txt”, kiiras);

Pl:
```
Random rnd = new Random();
string[] adatok = new string[10];
for (int i = 0; i < adatok.Length; i++)
  {
      adatok[i] = Convert.ToString(rnd.Next(1, 100));
  }
string kiiras = string.Join(":", adatok);
File.WriteAllText("proba.txt", kiiras);
```

2. **Több sornyi adat kiírása:**

Több sornyi adat kiírásakor szükségünk lesz egy szöveg típusú tömbre (kiir), amelybe tároljuk a kiírandó sorokat.
File.WriteAllLines(”filenév elérési útvonallal”, szöveg típusú tömb);
Pl.:
string[] kiir=new string{”alma”, ”körte”, ”szilva”};
File.WriteAllLines(”gyumolcsok.txt”, kiir);

3. **Több sor, soronként több adat**

Ebben az esetben az előző két rész megoldásait kell ötvöznünk. Először biztosítani kell, hogy az elemek a megfelelő elválasztó karakterrel együtt egy sorba kerüljenek. Majd ezeket a sorokat egy sztring típusú tömbben tároljuk. Itt is hasznos lehet a string.Join(" ", sztring típusú tömb) megoldás.
Pl.: Írjunk ki 5 sorba, soronként 3 darab 1 és 100 közé eső véletlenszámot * karakterrel el elválasztva a *veletlenek.txt* fájlba!

```
Random rnd = new Random();
string[] egy_sornyi = new string[3];
string[] kiir = new string[5];
for (int i = 0; i < kiir.Length; i++)
{
   for (int j = 0; j < egy_sornyi.Length; j++)
   {
      egy_sornyi[j] = Convert.ToString(rnd.Next(1, 101));          //véletlenek sztringgé alakítása
   }
   kiir[i] = string.Join("*", egy_sornyi);      // egy sornyi adat sztringgé alakítása
}
File.WriteAllLines("veletlenek.txt", kiir);     // a sztring típusú tömb kiírása
```