OOP III.
A C# nyelv alapelemei

2. rész

Alaptípusok (2. rész)
Operátorok és precedenciájuk (2. rész)
Érték- és referenciatípusok
Utasítások: for, foreach, continue, return, goto
Műveletek karaktersorozatokkal

Készítette:
Dr. Kotsis Domokos
Miklós Árpád

Hallgatói tájékoztató

A jelen bemutatóban található adatok, tudnivalók és információk a számonkérendő anyag vázlatát képezik. Ismeretük szükséges, de nem elégséges feltétele a sikeres zárthelyinek, illetve vizsgának.

Sikeres zárthelyihez, illetve vizsgához a jelen bemutató tartalmán felül a kötelező irodalomként megjelölt anyag, a gyakorlatokon szóban, illetve a táblán átadott tudnivalók ismerete, valamint a gyakorlatokon megoldott példák és az otthoni feldolgozás céljából kiadott feladatok önálló megoldásának képessége is szükséges.

A C# beépített alaptípusai (2)

- **Egész számok (2)**

Név	Leírás	Értéktartomány
sbyte	8 bites előjeles egész	-128 : 127
byte	8 bites előjel nélküli egész	0 : 255
short	16 bites előjeles egész	-32 768 : 32 767
ushort	16 bites előjel nélküli egész	0 : 65535
long	64 bites előjeles egész	-9 223 372 036 854 775 808 : 9 223 372 036 854 775 807
ulong	64 bites előjel nélküli egész	0 : 18 446 744 073 709 551 615

- **Valós számok**

Név	Leírás	Értékes jegy	Értéktartomány
float	32 bites lebegőpontos	7	$\pm 1{,}5*10^{-45}$: $\pm 3{,}4*10^{38}$
double	64 bites lebegőpontos	15	$\pm 5{,}0*10^{-324}$: $\pm 1{,}7*10^{308}$
decimal	128 bites nagypontosságú	28	$\pm 1{,}0*10^{-28}$: $\pm 7{,}9*10^{28}$

Valós számok gépi ábrázolása

- **Bináris (kettes számrendszerbeli) számábrázolás**
  - Tárolásuk 0 és 1 értékű számjegyek (bitek) sorozataként történik
- **Ábrázolás: ún. lebegőpontos („floating point”) forma**

$$\text{előjel} * (1 + \text{törtrész}) * 2^{\text{kitevő-eltolás}}$$

IEEE-754 szabvány	Méret	Előjel	Kitevő	Törtrész	Eltolás
Egyszeres pontosság	32 bit	1 bit	8 bit	23 bit	127
Kétszeres pontosság	64 bit	1 bit	11 bit	52 bit	1023

  - **Előjel**: 0 jelöli a pozitív, 1 a negatív számokat
  - **Kitevő**: 1-nél kisebb és nagyobb számokat is szeretnénk ábrázolni, ezért adott nagyságú **eltolás** alkalmazásával tároljuk a kitevőt
  - **Törtrész**: a „kettedespont” utáni számjegyeket tartalmazza
    - Az **(1+törtrész)** tag neve **mantissza**
    - Optimalizálás: a mantissza egész részét (az 1-et) nem tároljuk, mivel tudjuk, hogy a mantissza értéke mindig 1 és 2 között van – így kétszeres az ábrázolási tartomány

Valós számok gépi ábrázolása

- **Speciális számok ábrázolása**
  - 0
    - Megállapodás szerint ha a kitevő és a törtrész csupa 0, maga a szám is 0
    - Külön +0 és -0 ábrázolható, de ezek egyenértékűek
  - ±∞
    - Megállapodás szerint ha a kitevő csupa 1, a törtrész csupa 0, akkor a szám ±∞
    - A végtelen elfogadott, bizonyos műveletekhez használható érték (!)
- **Nem teljes pontosságú számábrázolás**
  - A hatvány formában történő tárolás miatt az utolsó értékes számjegyek elvesznek
  - A kettes számrendszerbeli ábrázolás következtében a végtelen „kettedestörtek” pontatlanságot okoznak
  - A fentiek miatt a lebegőpontos ábrázolás a valós számok meghatározott részhalmazát képes csak ábrázolni
- **Nagy ábrázolható számtartomány**

Gyakorló feladat - Pontosság I.

CS 1
Készítsünk programot, mely egy double típusú lebegőpontos változó értékét 0-tól 1 tizedenként növeli, míg csak az érték 100 nem lesz. A növelést végezze ciklusban.
A ciklus magjában helyezzen el vizsgálatot, mely kiírja a változó értékét, ha az nagyobb lesz, mint 200(!).
Lesz ilyen kiírás?

(Egy ciklusból a `break` paranccsal léphetünk ki.)

a = 200.10!!!

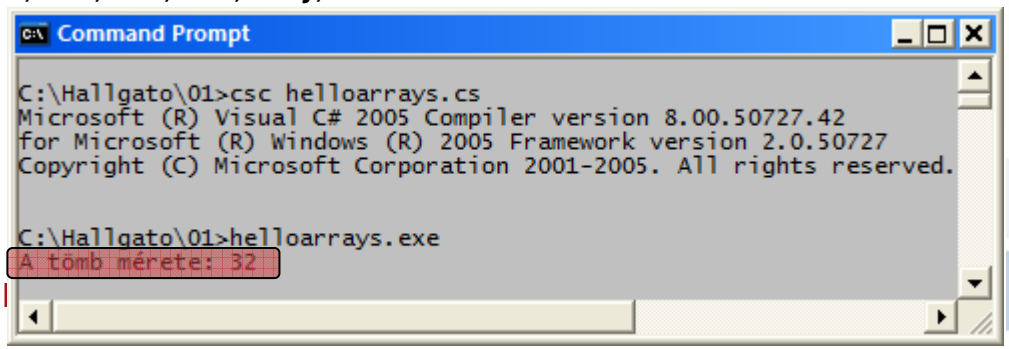
Pontosság II.

```
class Double
{
        static void Main()
        {
                double a;
                a = 0;
                do
                {
                        a = a + 0.1;
                        if (a > 200)
                        {
                                System.Console.WriteLine("a ="+a);
                                break;
                        }
                }
                while (a != 100);
                System.Console.ReadLine();
        }
}
```

Egyéb alaptípusok: a tömb (1)

- **A tömbök („array”) adattípusa bármilyen beépített típus vagy újonnan definiált saját típus lehet**
- **A tömbök indexelése 0-tól kezdődik**
- **A tömbök ún. referenciatípusok**
  - Később részletesebben tárgyaljuk

```
class HarmadikProgram
{
    static void Main()
    {
        int[] egésztömb = new int[32];
        int[] ElőreMegadottTömb = {2, 3, 5, 7, 11, 13, 17, 19};

        egésztömb[20] = 1;
        egésztömb[31] = 9;

        int tömbméret = egésztömb.Length;
        System.Console.WriteLine("A tömb
    }
}
```

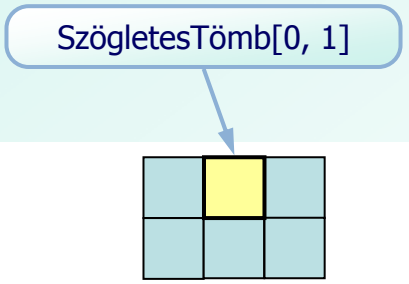


helloarrays.cs

Egyéb alaptípusok: a tömb (2)

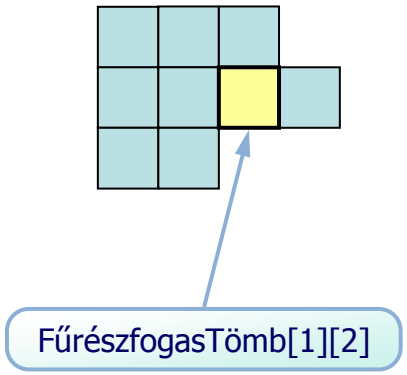
- **Többdimenziós tömbök**
  - „Szögletes” tömbök („rectangular array”)

```
int[,] SzögletesTömb = new int[2, 3];

int x = SzögletesTömb[0, 1];
System.Console.WriteLine(SzögletesTömb.Length);
System.Console.WriteLine(SzögletesTömb.GetLength(1));
```



  - „Fűrészfogas” tömbök („jagged array”)

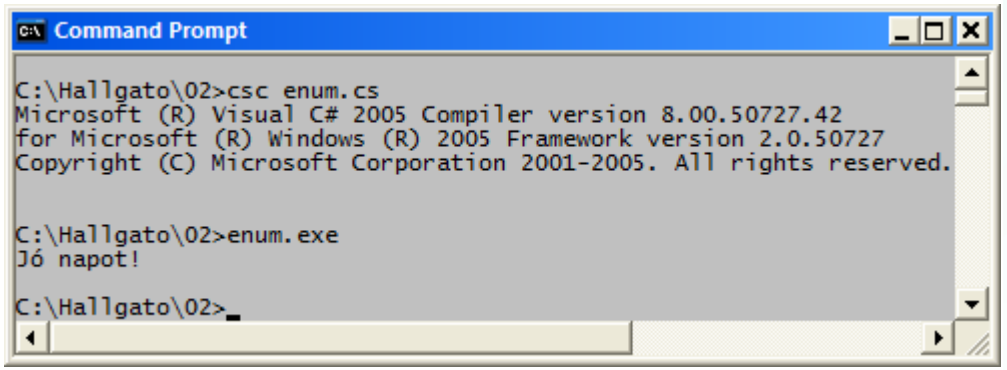
```
int[][] FűrészfogasTömb = new int[3][];

FűrészfogasTömb[0] = new int[3];
FűrészfogasTömb[1] = new int[4];
FűrészfogasTömb[2] = new int[2];

int y = FűrészfogasTömb[0][1];
System.Console.WriteLine(FűrészfogasTömb.Length);
System.Console.WriteLine(FűrészfogasTömb[1].Length);
```



Egyéb alaptípusok: a felsorolás

- **A felsorolás a programozó által definiált egész típus, amely csak a megadott értékeket veheti fel**
  - A felsorolás egyes érvényes értékeinek külön név is adható

```
enum Napszak
{
    Reggel = 0,
    Délelőtt = 1,
    Este = 4
}
...
Napszak időpont = Napszak.Este;
switch (időpont)
{
    case Napszak.Reggel:
        System.Console.WriteLine("Jó reggelt!");
        break;
    default:
        System.Console.WriteLine("Jó napot!");
        break;
}
```



enum.cs

Érték- és referenciatípusok

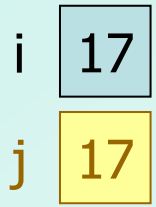
- **A C# kétféle adattípust különböztet meg: értéktípusokat („value type”), illetve referenciatípusokat („reference type”)**
- **Az értéktípusok közvetlenül tárolják adataikat**
  - Értékül adásnál az adatok új másolata jön létre
    - Az értékadást követően a két adatpéldány teljesen függetlenül viselkedik
  - Paraméterátadásnál az adatokat szintén mindig át kell másolni
- **A referenciatípusok csak hivatkozást tárolnak az adatokra**
  - Értékül adásnál csak az adatokra való hivatkozás másolódik le
    - Az értékadást követően az új adatpéldány fizikailag azonos marad a régivel, mindössze egy új hivatkozás keletkezik
  - Paraméterátadásnál nem kell átmásolni az adatokat
- **Később mindkét kategóriát részletesebben tárgyaljuk**

Érték- és referenciatípusok (példa)

1.

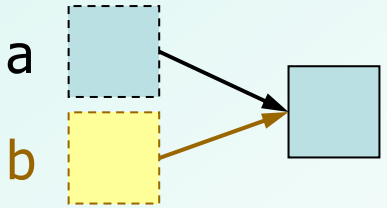
```
int i = 17;
int j = i;
```

Az „int” típus **értéktípus**




```
int[] a = new int[1];
int[] b = a;
```

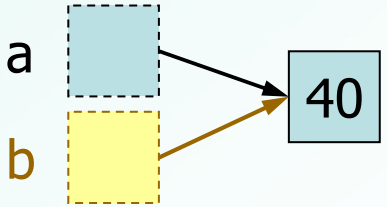
A tömbök **referenciatípus**ok



2.

```
i = 23;
j = 36;
```


```
a[0] = 15;
b[0] = 40;
```



3.

```
System.Console.WriteLine(i);    // 23

System.Console.WriteLine(j);    // 36
```

```
System.Console.WriteLine(a[0]);    // 40

System.Console.WriteLine(b[0]);    // 40
```

Érték- és referenciatípusok

- **A C# teljes típusrendszerének összefoglalása**

Kategória	Alkategória	Leírás
Érték-típusok	Egyszerű típusok	Előjeles és előjel nélküli egész számok (sbyte, short, int, long, byte, ushort, uint, ulong)
		Unicode karakterek (char)
		Valós számok (float, double; decimal)
		Logikai adattípus (bool)
	Felsorolások	Programozó által definiált enum típusok
	Struktúrák*	Programozó által definiált struct típusok*
Referencia-típusok	Osztályok*	Az object ősosztály*
		Unicode karaktersorozatok (a string osztály)*
		Programozó által definiált osztályok (class)*
	Interfészek*	Programozó által definiált interface típusok*
	Tömbök	Egy- és többdimenziós tömbök (array)
	Képviselők*	Programozó által definiált delegate típusok*

Operátorok és precedenciájuk (2)

- **Hozzáférési célú operátorok**

Operátor	Kifejezés	Precedencia	Jelentés
.	x.y	1	Taghozzáférés (összetett típusoknál és felsorolásoknál)
( )	f(x)	1	Metódushívás (tagfüggvények végrehajtása egyes összetett típusoknál)
[ ]	a[x]	1	Tömbelem-hozzáférés (tömböknél), hozzáférés indexelt tulajdonsághoz*

- **Egyéb operátorok**

Operátor	Kifejezés	Precedencia	Jelentés
? :	x ? y : z	13	Ha az „x” feltétel igaz, akkor a kifejezés értéke „y”, ellenkező esetben „z” lesz

Konverzió, cast-olás

```
float floatszám;
int intszám;
double doubleszám;
string s;
...
...
...
floatszám = System.Convert.ToSingle(s);
floatszám = float.Parse(s);
intszám = System.Convert.ToInt32(s);
intszám = int.Parse(s);
s = intszám.ToString();
s = floatszám.ToString();
floatszám = (float)doubleszám;
```

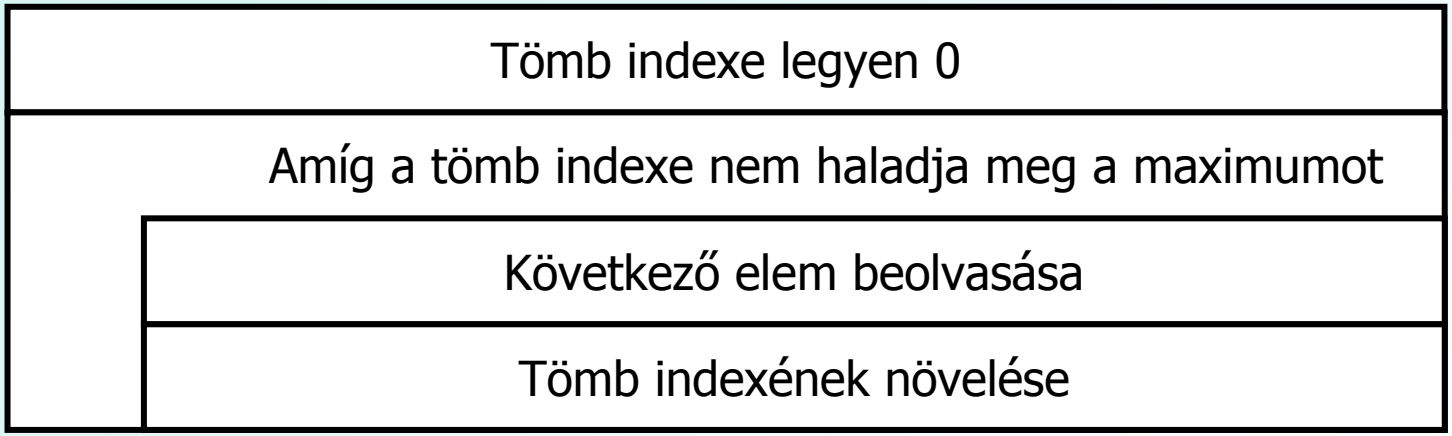
Gyakorló feladatok

CS 2
Készítsünk programot, amely a konzolról beolvas egy nevet és egy születési évet, majd kiírja az illető korát!

```
class Életkor
{
    static void Main()
    {
        int évszám, életkor;
        string név;
        System.Console.WriteLine("Név: ");
        név = System.Console.ReadLine();
        System.Console.WriteLine("Születés éve: ");
        évszám = System.Convert.ToInt32(System.Console.ReadLine());

        életkor = 2011 - évszám;

        System.Console.WriteLine(név + " életkora: " + életkor + " év");
        System.Console.ReadLine();
    }
}
```

Gyakorló feladatok

CS 3
Készítsünk struktogram formájában algoritmust, amely elvégzi egy egydimenziós tömb feltöltését a konzolról beolvasott adatokkal!

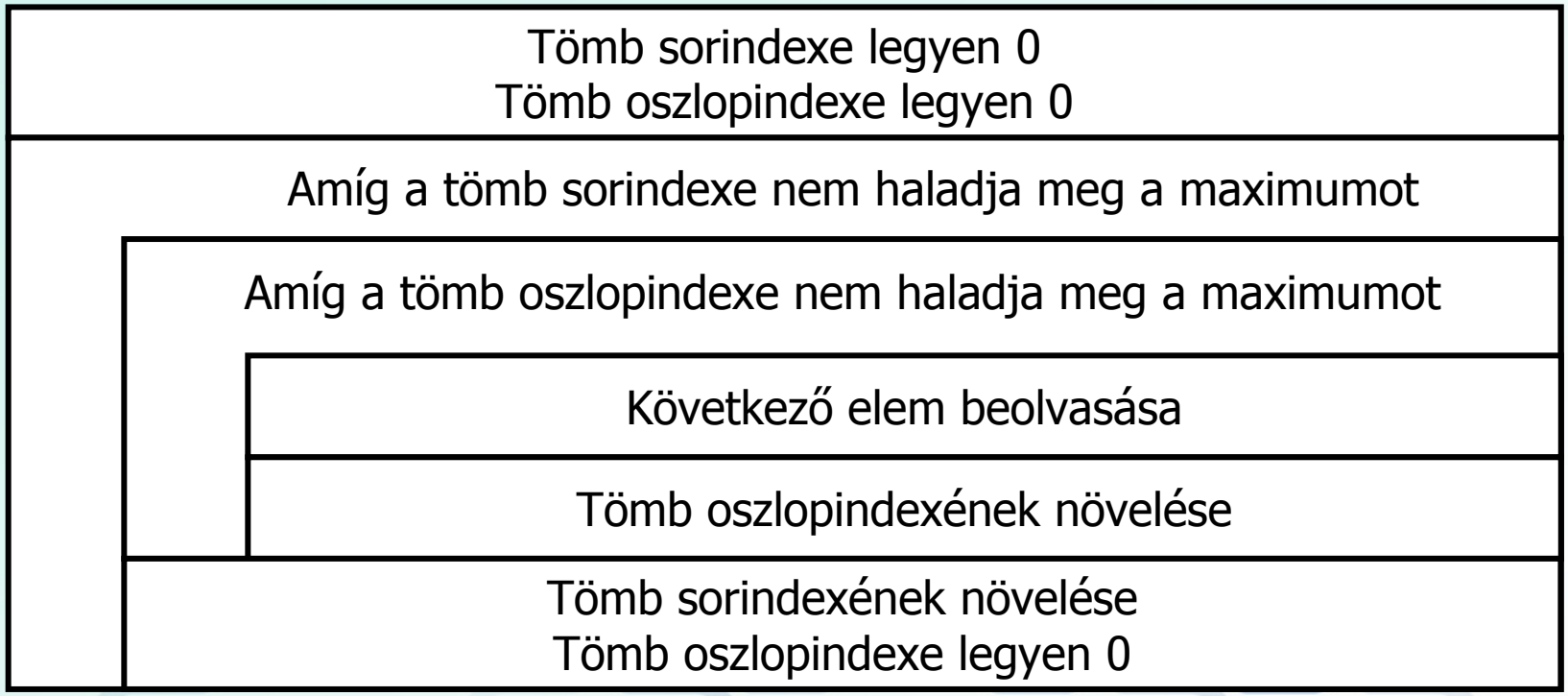


Gyakorló feladatok

CS 4
Készítsünk algoritmust, majd programot, amely a konzolról beolvassa egy kétdimenziós, 3x3-as tömb minden elemét, majd kiírja a tömb teljes tartalmát!

Gyakorló feladatok

CS 4 – az algoritmus struktogramja



Gyakorló feladatok

CS 4 – a program 1. része

```
class Tömbkezelő
{
    static void Main()
    {
        string[,] egésztömb = new string[3, 3];  int i = 0, j = 0;  string s;
        while (i <= 2)
        {
                System.Console.WriteLine("A(z) " + (i+1) + ". sor:");
                while (j <= 2)
                {
                        System.Console.WriteLine("A(z) " + (j+1) + ". elem:");
                        s = System.Console.ReadLine();
                        egésztömb[i, j] = s;
                        j++;
                }
                i = i + 1;
                j = 0;
        }
```

...

Gyakorló feladatok

CS 4 – a program 2. része

```
...
        i = 0; j = 0;
        while (i <= 2)
        {
                System.Console.WriteLine("A(z) " + (i+1) + ". sor tartalma:");
                while (j <= 2)
                {
                        System.Console.Write(egésztömb[i, j]);
                        System.Console.Write("    ");
                        j++;
                }
                i++;
                j = 0;
                System.Console.WriteLine();
        }
        System.Console.ReadLine();
    } // Main() vége
} // Tömbkezelő vége
```

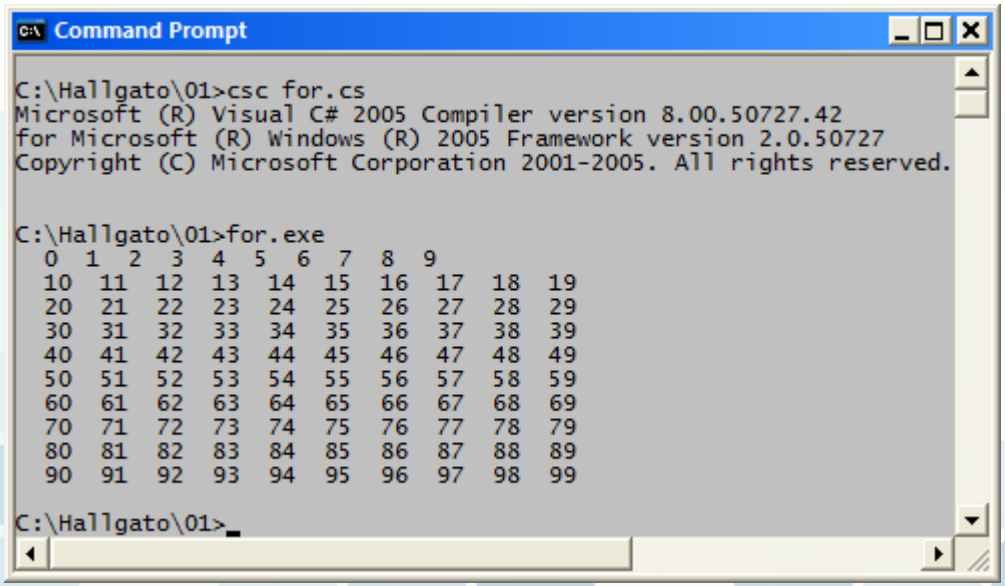
A for utasítás

```
for (inicializátor; feltétel; iterátor)
    utasítás
```

- **Az inicializátor és az iterátor tetszőleges utasítás lehet**
- **Működése:**
  - Belépéskor egyszer végrehajtódik az inicializátor
  - Minden ciklusmenetben kiértékelődik a feltétel
  - Amennyiben a feltétel igaz, az utasítás (a „ciklusmag”) egyszer lefut
  - A ciklusmag végeztével végrehajtódik az iterátor és ismét kiértékelődik a feltétel
  - A ciklus akkor ér véget, amikor a feltétel hamissá válik, ellenkező esetben újabb ciklusmenet következik
- **Általában az inicializátor egy számlálót állít be, az iterátor pedig ezt a számlálót növeli vagy csökkenti**
  - Legtöbbször akkor használjuk, ha előre ismert számú alkalommal szeretnénk végrehajtani egy utasítást

A for utasítás (példa)

```
// Számmátrix

// Ez a külső ciklus fut végig az összes soron
for (int i = 0; i < 100; i += 10)
{
    // Ez a belső ciklus fut végig egy soron belül az összes oszlopon
    for (int j = i; j < i + 10; j++)
    {
        System.Console.Write("  " + j);
    }
    System.Console.WriteLine();
}
```



for.cs

Gyakorló feladatok

CS 5
Készítsünk algoritmust, majd programot, amely a konzolról beolvassa egy kétdimenziós, 3x3-as tömb minden elemét, majd kiírja a tömb teljes tartalmát! Használjuk a for utasítást!

A foreach utasítás

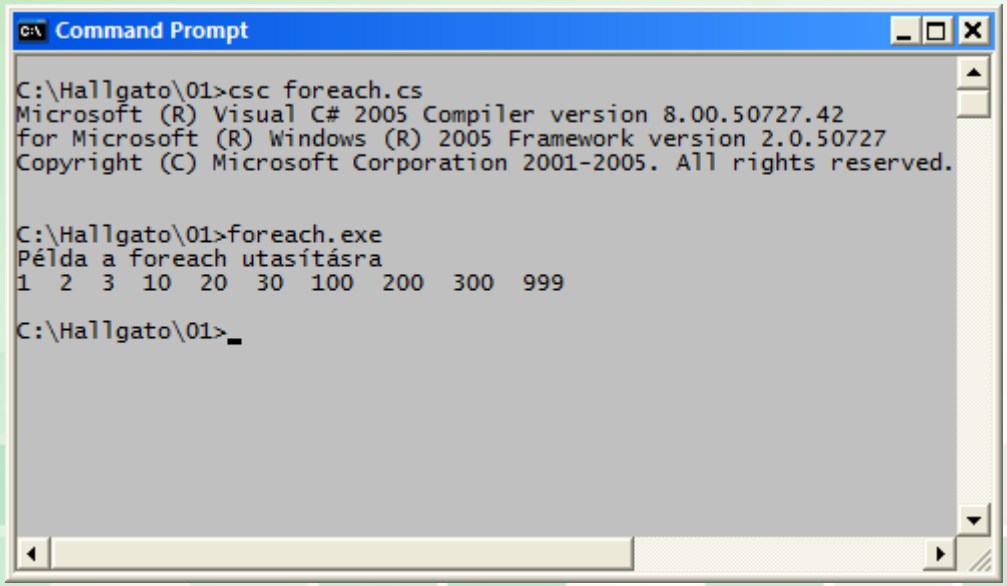
```
foreach (típus változó in gyűjtemény)
    utasítás
```

- **Lehetővé teszi egy utasítás végrehajtását egy adott gyűjtemény összes elemére**
  - A „gyűjtemény” pontos fogalmát később részletesen tárgyaljuk
  - A tömbök gyűjtemények, tehát a `foreach` utasítás használható hozzájuk
- **Működése:**
  - Belépéskor létrejön egy „típus” típusú változó („iterációs változó”)
    - Ez a változó csak az utasításon belül használható
  - Az utasítás annyiszor hajtódik végre, ahány elemet tartalmaz a gyűjtemény
  - Az iterációs változó minden egyes végrehajtásnál felveszi a gyűjtemény soron következő elemének értékét
- **Az iterációs változó az utasításban nem módosítható**
  - Erre a célra a `for` utasítás használható

A foreach utasítás (példa)

```
int[] teszttömb = {1, 2, 3, 10, 20, 30, 100, 200, 300, 999};

System.Console.WriteLine("Példa a foreach utasításra");
foreach (int tömbérték in teszttömb)
{
    System.Console.Write(tömbérték + "  ");
}
System.Console.WriteLine();
```

foreach.cs

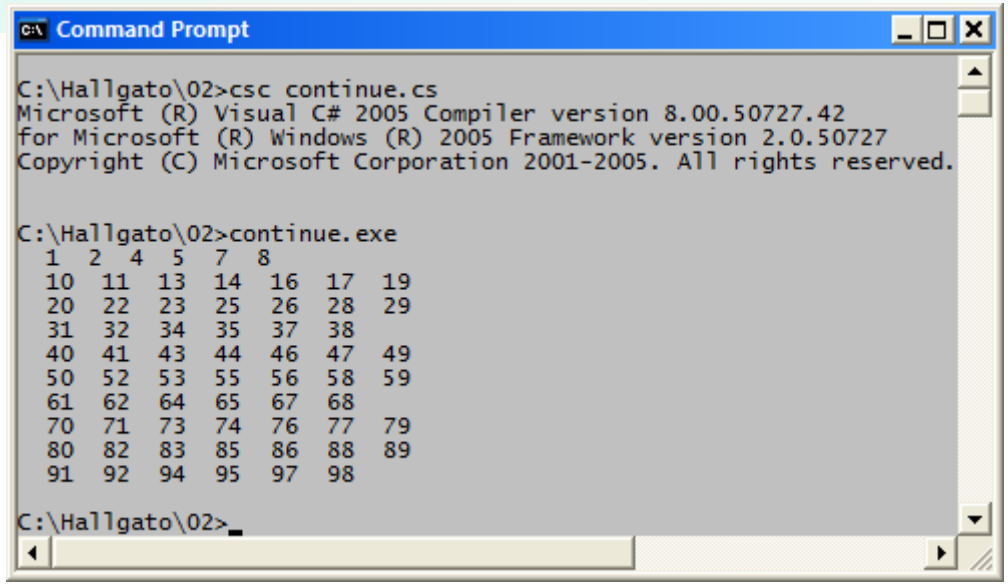


A continue utasítás

continue ;

- **Az aktuális ciklusmenet megszakítása, folytatás a következő ciklusmenettel**
  - Az aktuális while, do...while, for, illetve foreach utasítás ciklusmagjából hátralévő rész átlépésére és a következő ciklusmenettel történő folytatásra használhatjuk

```
for (int i = 0; i < 100; i += 10)
{
    for (int j = i; j < i + 10; j++)
    {
        // Kihagyjuk a hárommal oszthatókat
        if (j % 3 == 0)
          continue;
        System.Console.Write("  " + j);
    }
    System.Console.WriteLine();
}
```



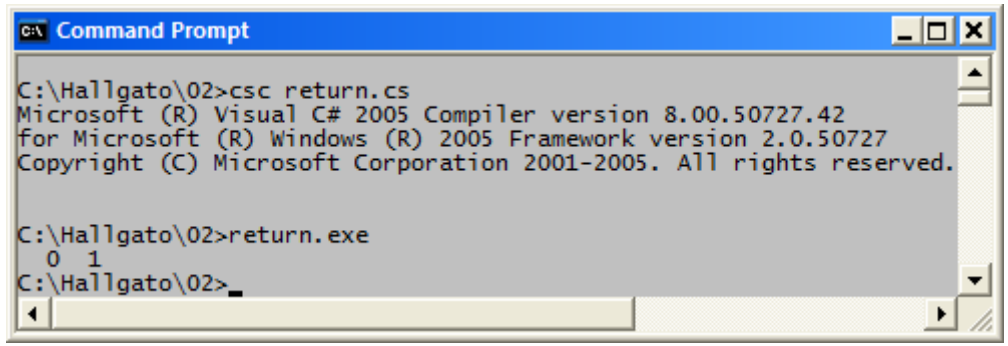
continue.cs

A return utasítás

```
return ;

return kifejezés;
```

- **A hátralévő utasítások átugrása és visszatérés a hívó utasításhoz**
  - A kifejezés értéke lesz a hívónak átadott visszatérési érték
    - Ha nincs visszatérési érték (azaz „void”), akkor nem adható meg kifejezés

```
for (int i = 0; i < 100; i += 10)
{
    for (int j = i; j < i + 10; j++)
    {
        System.Console.Write("  " + j);
        // Kilépés az első páratlan szám után
        if (j % 2 == 1)
            return;
    }
    System.Console.WriteLine();
}
```



return.cs

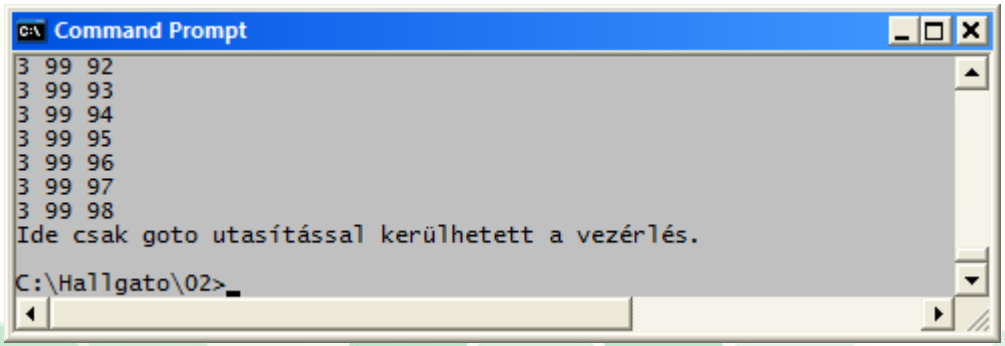
A goto utasítás

```
goto címke;

goto case címkekonstans;

goto default;
```

- **Közvetlen ugrás a megadott címkéhez**
  - Utasítás belsejébe nem lehet ilyen módon belépni
- **switch utasításnál ugrás a megadott konkrét (case), illetve alapértelmezett (default) címkéhez**
  - Ezzel az is elérhető, hogy a switch utasításnál több különböző esetben is végrehajtódjon ugyanaz az utasítássorozat (a megoldás neve „átesés”)
- **Használata általában nem javasolt**
  - Könnyen átláthatatlanná teheti a programvégrehajtás menetét
  - Rendszeres használata elavult, a strukturált programozás kora előtti stílusra utal

A goto utasítás (példa)

```
for (int i = 0; i < 100; i ++)
{
    for (int j = 0; j < 100; j++)
    {
        for (int k = 0; k < 100; k++)
        {
                if ( (i + j + k) > 200)
                        goto Probléma;
                System.Console.WriteLine(i + " " + j + " " + k);
        }
    }
}
return;

Probléma:
    System.Console.WriteLine("Ide csak goto utasítással kerülhetett a vezérlés.");
```



goto.cs

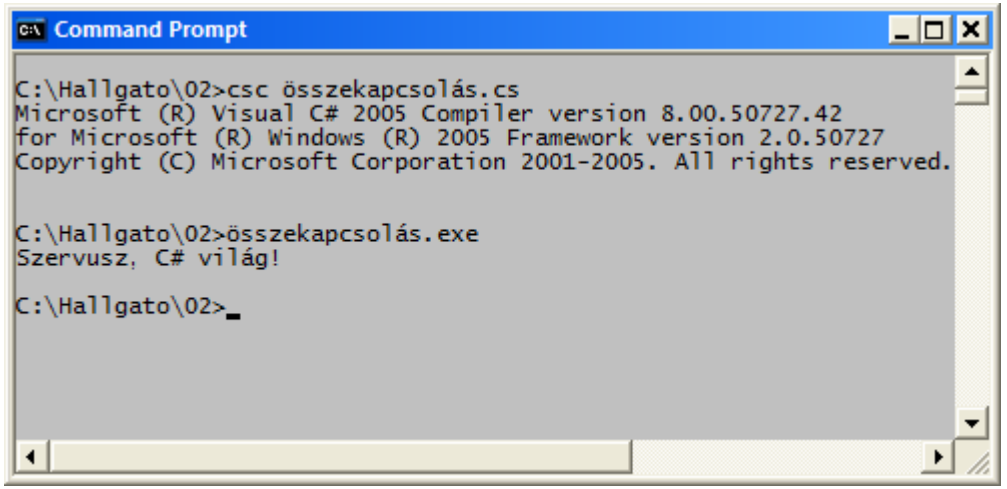
Műveletek karaktersorozatokkal (1)

- **A karaktersorozat („string”) karakter típusú elemekből álló egydimenziós tömbként is felfogható**
- **Mivel gyakran használt, igen fontos típusról van szó, rengeteg beépített segédfunkció áll rendelkezésre hozzá**
- **Néhány kiemelt művelet és segédfunkció:**
  - Összekapcsolás (+ operátor)
  - Részsorozat kiválasztása (**Substring** függvény)
  - Részsorozat keresése (**IndexOf** és **LastIndexOf** függvény)
  - Karaktersorozat átalakítása számmá (**Convert.ToInt32** függvény)
  - Szám átalakítása karaktersorozattá (**Convert.ToString** függvény)
  - Kis- és nagybetűs formára alakítás (**ToUpper, ToLower** függvények)
  - Formázott megjelenítés (**String.Format** függvény)
  - Karaktersorozat kezelése tömbként

Műveletek karaktersorozatokkal (2)

- **Összekapcsolás**

```
class Összekapcsolás
{
    static void Main()
    {
        string str1 = "Szervusz";
        string str2 = "C#";
        string str3 = "világ!";
        string str4 = str1 + ", " + str2 + " " + str3;
        System.Console.WriteLine(str4);
    }
}
```

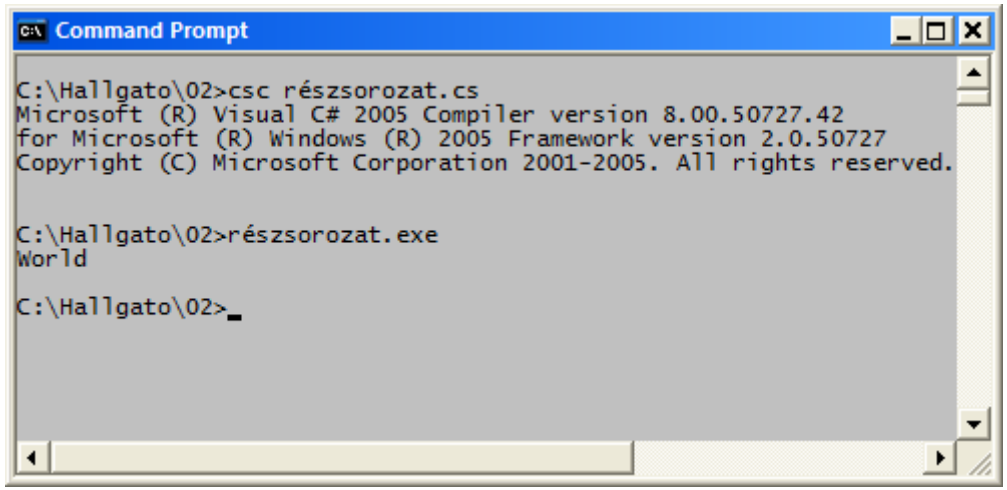


összekapcsolás.cs

Műveletek karaktersorozatokkal (3)

- **Részsorozat kiválasztása**

```
class Részsorozat
{
    static void Main()
    {
        string s1, s2;
        s1 = "Hello, World";
        s2 = s1.Substring(7, 5);
        System.Console.WriteLine(s2);
    }
}
```

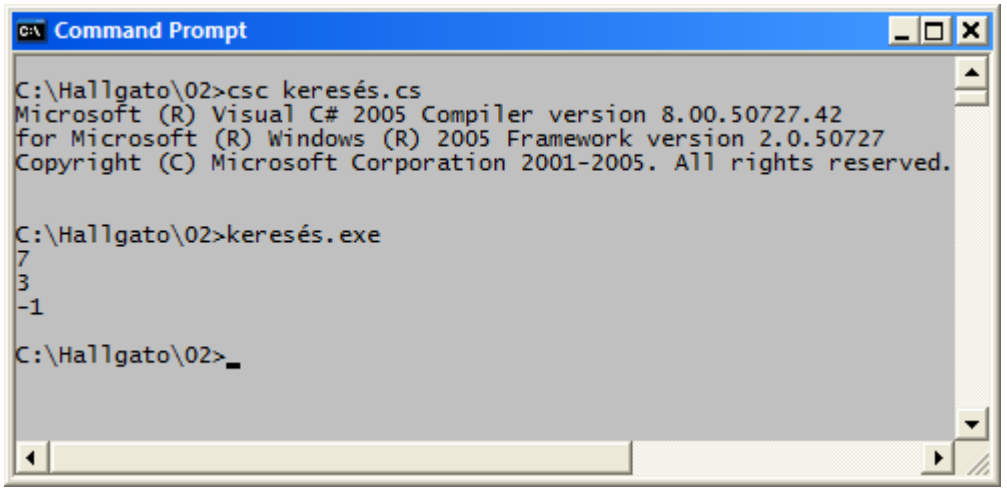


részsorozat.cs

Műveletek karaktersorozatokkal (4)

- **Részsorozat keresése**

```
class Keresés
{
    static void Main()
    {
        int i;
        string s1;
        s1 = "Ez egy karaktersorozat";
        i = s1.IndexOf("karakter");
        System.Console.WriteLine(i);
        i = s1.IndexOf("egy");
        System.Console.WriteLine(i);
        i = s1.IndexOf("ez nincs benne");
        System.Console.WriteLine(i);
    }
}
```

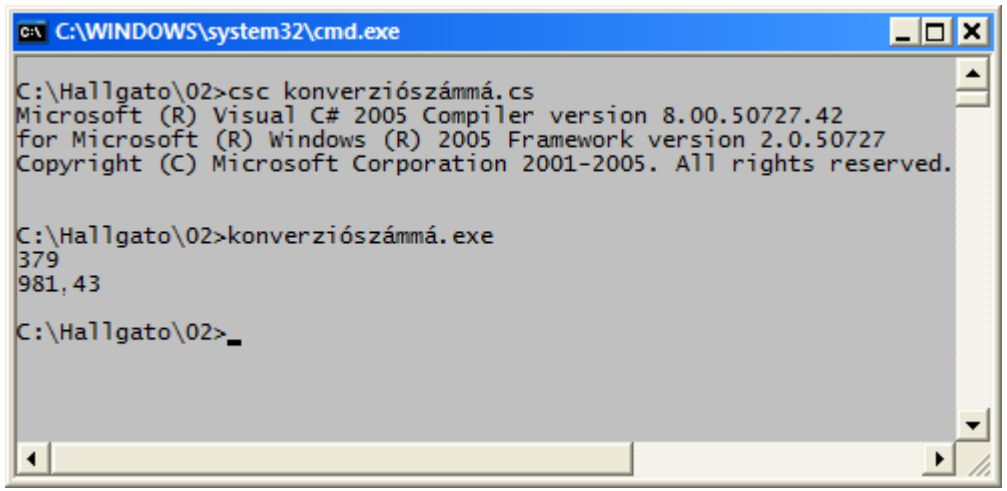


keresés.cs

Műveletek karaktersorozatokkal (5)

- **Karaktersorozat átalakítása számmá**

```
class KonverzióSzámmá
{
    static void Main()
    {
        int i, j, k;
        float f;
        string s1, s2, s3;
        s1 = "123";
        s2 = "256";
        s3 = "981,43";

        i = System.Convert.ToInt32(s1);
        j = System.Convert.ToInt32(s2);
        k = i + j;
        System.Console.WriteLine(k);

        f = float.Parse(s3);
        System.Console.WriteLine(f);
    }
}
```

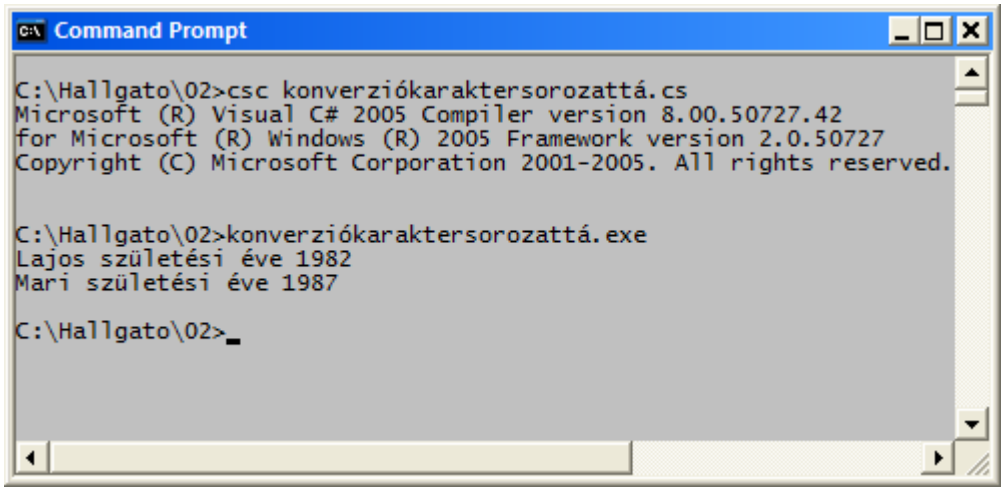


konverziószámmá.cs

Műveletek karaktersorozatokkal (6)

- **Szám átalakítása karaktersorozattá**

```
class KonverzióKaraktersorozattá
{
    static void Main()
    {
        int i, j;
        string s1, s2;

        i = 1982;
        j = 1987;
        s1 = "Lajos születési éve " + i;
        s2 = "Mari születési éve " + System.Convert.ToString(j);
        System.Console.WriteLine(s1);
        System.Console.WriteLine(s2);
    }
}
```

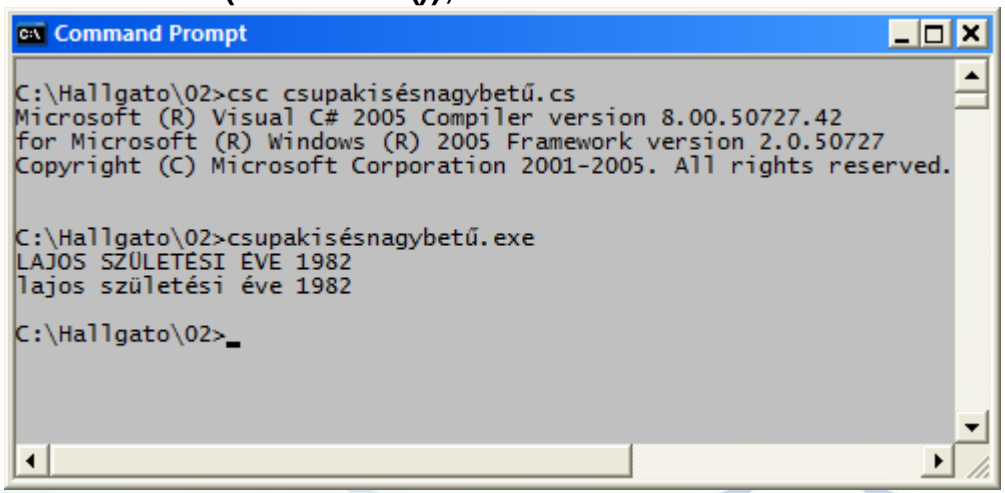


konverziókaraktersorozattá.cs

Műveletek karaktersorozatokkal (7)

- **Kis- és nagybetűs formára alakítás**

```
class CsupaKisÉsNagybetű
{
    static void Main()
    {
        int i;
        string s;

        i = 1982;
        s = "Lajos születési éve " + i;
        System.Console.WriteLine(s.ToUpper());
        System.Console.WriteLine(s.ToLower());
    }
}
```

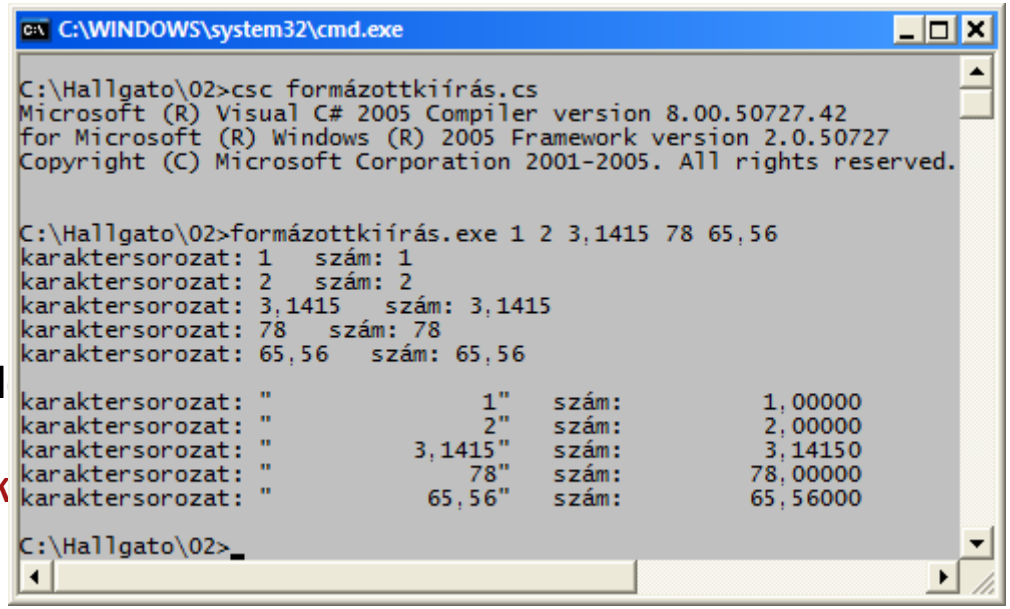


csupakisésnagybetű.cs

Műveletek karaktersorozatokkal (8)

- **Formázott megjelenítés parancssori paraméterekkel**

```
class FormázottMegjelenítés
{
    static void Main(string[] args)
    {
        float szám;
        foreach (string s in args)
        {
                szám = System.Convert.ToSingl
                System.Console.WriteLine(
                        System.String.Format("k
        }
        System.Console.WriteLine();
        foreach (string s in args)
        {
                szám = System.Convert.ToSingle(s);
                System.Console.WriteLine(
                        System.String.Format("karaktersorozat: \"{0, 16}\"   szám: {1, 16:N5}", s, szám));
        }
    }
}
```



formázottmegjelenítés.cs

Műveletek karaktersorozatokkal (8)

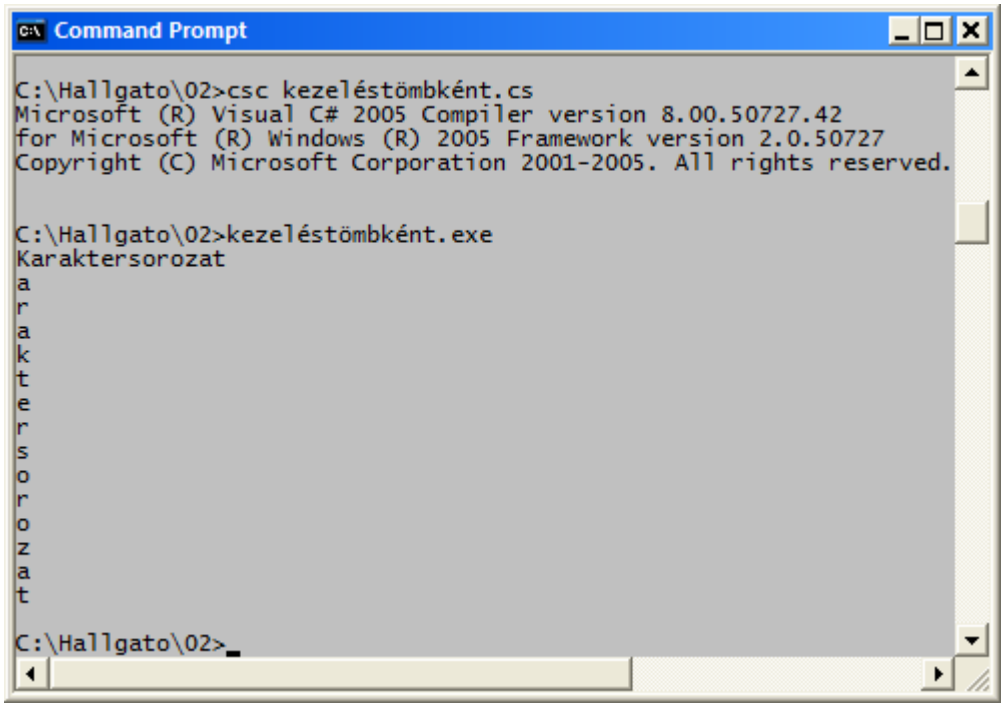
- **Formázott megjelenítés vezérlőkarakterei**

Kód	Számtípus	Magyarázat	Példa
C	Egész és valós	Helyi pénznem formázási szabályai szerinti kijelzés	1 435,5 Ft (Magyarország) $1435.5 (USA)
D	Csak egész	Általános egész szám	1435
E	Egész és valós	Tudományos jelölésmód	1,4355E+003 (Magyarország) 1.4355E+003 (USA)
F	Egész és valós	Fixpontos decimális számkijelzés	1435,50 (Magyarország) 1435.50 (USA)
G	Egész és valós	Általános számkijelzés	1435,5 (Magyarország) 1435.5 (USA)
N	Egész és valós	Helyi területi beállítások szerinti számkijelzés	1 435,500 (Magyarország) 1,435.500 (USA)
P	Egész és valós	Százalékos formátum	143 550,00 %
X	Csak egész	Hexadecimális formátum	59B

Műveletek karaktersorozatokkal (9)

- **Karaktersorozat kezelése tömbként**

```
class KezelésTömbként
{
    static void Main()
    {
        int i;
        string s;
        s = "Karaktersorozat";

        foreach (char c in s)
                System.Console.Write(c);
        System.Console.WriteLine();

        i = 1;
        while (i < s.Length)
        {
                System.Console.WriteLine(s[i]);
                i++;
        }
    }
}
```



kezeléstömbként.cs

Gyakorló feladatok

CS 6
Készítsen programot, amely egy stringben megkeresi egy adott karakter valamennyi előfordulását!

Gyakorló feladatok

CS 7
Készítsen programot, amely egy stringben kicserél minden *A* karaktert *B*-re!

Gyakorló feladatok

CS 8
Készítsen programot, amely egy adott karaktersorozatot (pl. „Amelyik kutya ugat, az a kutya nem harap”) minden adott karaktersorozatát (pl. „kutya”) egy adott karaktersorozatra (pl. „macska”) cseréli!

Gyakorló feladatok

CS 9
Készítsünk programot háromelemű valós vektorok kezelésére az alábbi funkciókkal:

- **Beolvasás és kiírás (legfeljebb 10 vektor)**
- **Skaláris szorzat kiszámítása**
- **Vektor szorzása skalárral**

Irodalom, feladatok

- Kotsis-Légrádi-Nagy-Szénási: Többnyelvű programozástechnika, PANEM, Budapest, 2007
- Faraz Rasheed: C# School, Synchron Data, 2006 http://www.programmersheaven.com/2/CSharpBook
- Reiter István: C# jegyzet, DevPortal, 2010, http://devportal.hu/content/CSharpjegyzet.aspx