C# nyelv

Története

- A C# (ejtsd: szí-sárp) a Visual Basic mellett a .NET fő programozási nyelve.
- 1999 –ben Anders Hejlsberg vezetésével kezdték meg a fejlesztését.
- A C# tisztán objektumorientált, típus biztos, általános felhasználású nyelv.
- A nyelv elméletileg platform független (létezik Linux és Mac fordító is), de napjainkban a legnagyobb hatékonyságot a Microsoft implementációja biztosítja.

A C# szintaktikája

- a C programozási nyelv szintaxisát veszi alapul, ez három fontos szabályt von maga után:
  - Az egyes utasítások végén pontosvessző (;) áll
  - A kis- és nagybetűk különböző jelentőséggel bírnak, azaz a “program” és “Program” azonosítók különböznek
  - A program egységeit (osztályok, metódusok, stb.) ún. blokkokkal jelöljük ki, kapcsos zárójelek ({ és }) segítségével.

Kulcsszavak

- Szinte minden programnyelv definiál kulcsszavakat, amelyek speciális jelentőséggel bírnak a fordító számára.
- Ezeket az azonosítókat a saját meghatározott jelentésükön kívül nem lehet másra használni, ellenkező esetben a fordító hibát jelez.

A C# 77 kulcsszót ismer:

abstract	default	foreach	object	Sizeof	unsafe
as	delegate	goto	operator	stackalloc	ushort
base	do	If	out	Static	using
bool	double	implicit	override	String	virtual
break	else	In	params	Struct	volatile
byte	enum	int	private	Switch	void
case	event	interface	protected	This	while
catch	explicit	internal	public	Throw	
char	extern	Is	readonly	True	
checked	false	lock	ref	Try	
class	finally	long	return	Typeof	
const	fixed	namespace	sbyte	Uint	
continue	float	new	sealed	Ulong	
decimal	for	null	short	unchecked	

Ezeken kívül létezik még 23 azonosító, amelyeket a nyelv nem tart fenn speciális használatra, de különleges jelentéssel bírnak. Amennyiben lehetséges, kerüljük a használatukat “hagyományos” változók, metódusok, osztályok létrehozásánál:

add	equals	group	let	Remove	var
ascending	from	in	on	Select	where
by	get	into	orderby	Set	yield
descending	global	join	partial	Value	

Megjegyzések (kommentek)

A forráskódba megjegyzéseket tehetünk

- üzeneteket hagyhatunk (pl. egy metódus leírása) magunknak vagy a többi fejlesztőnek,
- dokumentációt tudunk generálni
- A kommenteket a fordító nem veszi figyelembe, tulajdonképpen a fordítóprogram első lépése, hogy a forráskódból eltávolít minden megjegyzést

```
using System;

class HelloWorld
{
      static public void Main()
      {
            Console.WriteLine("Hello C#"); // Ez egy egysoros komment
            Console.ReadKey();
            /* Ez
                  egy
             többsoros komment */
      }
}
```

Változók

- Változók: tárolók, ahová az adatainkat ideiglenesen eltároljuk
- a memória egy (vagy több) cellájára hivatkozó leírók

Egy változót a következő módon hozhatunk létre C# nyelven:

Típus változónév;

Pl: string i;

Változónévre vonatkozó szabályok

- A változónév első karaktere csak betű vagy alulvonás jel (_) lehet, a többi karakter szám is.
- Lehetőleg kerüljük az ékezetes karakterek használatát.
- Konvenció szerint a változónevek kisbetűvel kezdődnek.
- Amennyiben a változónév több szóból áll, akkor célszerű azokat a szóhatárnál nagybetűvel “elválasztani” (pl. pirosAlma, vanSapkaRajta, stb.).

Típusok

- A C# erősen (statikusan) típusos nyelv
- minden egyes változó típusának ismertnek kell lennie fordítási időben, ezzel biztosítva azt, hogy a program pontosan csak olyan műveletet hajthat végre amire valóban képes.
- A típus határozza meg, hogy egy változó milyen értékeket tartalmazhat, illetve mekkora helyet foglal a memóriában.

C# típus	.NET típus	Méret (byte)	Leírás
byte	*System.Byte*	1	Előjel nélküli 8 bites egész szám (0..255)
char	*System.Char*	2	Egy Unicode karakter
bool	*System.Boolean*	1	Logikai típus, értéke igaz(1 vagy true) vagy hamis(0 vagy false)
sbyte	*System.SByte*	1	Előjeles, 8 bites egész szám (-128..127)
short	*System.Int16*	2	Előjeles, 16 bites egész szám (-32768..32767
ushort	*System.Uint16*	2	Előjel nélküli, 16 bites egész szám (0..65535)
int	*System.Int32*	4	Előjeles, 32 bites egész szám (–2147483648.. 2147483647).
uint	*System.Uint32*	4	Előjel nélküli, 32 bites egész szám (0..4294967295)
float	*System.Single*	4	Egyszeres pontosságú lebegőpontos szám
double	*System.Double*	8	Kétszeres pontosságú lebegőpontos szám
decimal	*System.Decimal*	16	Fix pontosságú 28+1 jegyű szám
long	*System.Int64*	8	Előjeles, 64 bites egész szám
ulong	*System.Uint64*	8	Előjel nélküli, 64 bites egész szám
string	*System.String*	N/A	Unicode karakterek szekvenciája
object	*System.Object*	N/A	Minden más típus őse

Konstansok

- A const típusmódosító kulcsszó segítségével egy objektumot konstanssá, megváltoztathatatlanná tehetünk.
- A konstansoknak egyetlen egyszer adhatunk (és ekkor kötelező is adnunk) értéket, mégpedig a deklarációnál. Bármely későbbi próbálkozás fordítási hibát okoz.
- A konstans változóknak adott értéket/kifejezést fordítási időben ki kell tudnia értékelni a fordítónak

```
const int x; // Hiba
const int x = 10; // Ez jó
x = 11; // Hiba
```

A konstans változóknak adott értéket/kifejezést fordítási időben ki kell tudnia értékelni a fordítónak. A következő forráskód éppen ezért nem is fog lefordulni:

```
using System;

class Program
{
      static public void Main()
      {
            Console.WriteLine("Adjon meg egy számot: ");
            const int x = int.Parse(Console.ReadLine());
      }
}
```

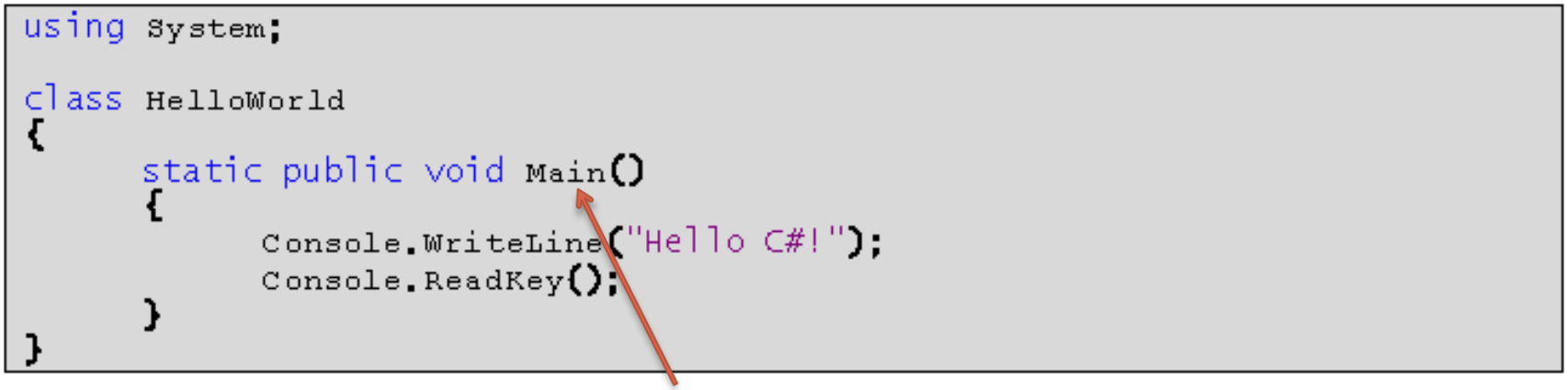
Osztályok

- Osztály: olyan adatok és műveletek összessége, amellyel leírhatjuk egy modell (vagy entitás) tulajdonságait és működését
- Legyen például a modellünk a kutya állatfaj.
- Egy kutyának vannak tulajdonságai (pl. életkor, súly, stb.)
- van meghatározott viselkedése (pl. csóválja a farkát, játszik, stb.)
- Amikor programot írunk, akkor az adott osztályból létre kell hoznunk egy (vagy több) példányt, ezt pédányosításnak nevezzük

Osztályok

- A tulajdonságokat tároló változókat **adattagoknak** (vagy mezőnek),
- a műveleteket **metódusoknak** nevezzük.
- A műveletek összességét **felületnek** is hívjuk.

A program szerkezete



```
using System;

class HelloWorld
{
      static public void Main()
      {
            Console.WriteLine("Hello C#!");
            Console.ReadKey();
      }
}
```

Az osztályok egyikében kell egy *Main nevű függvénynek szerepelnie, amely* futtatáskor az operációs rendszertől a vezérlést megkapja

Input, Output

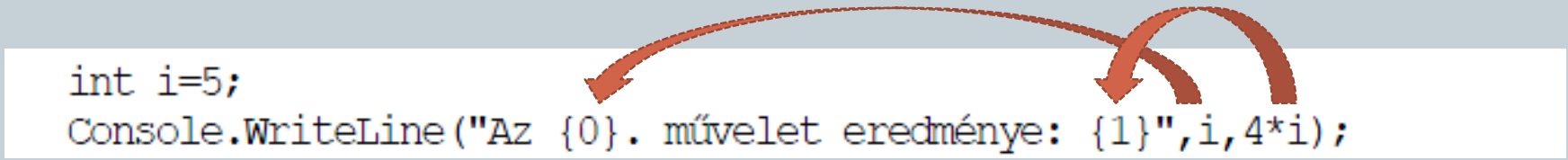
- Minden klasszikus konzolprogram végrehajtása esetén automatikusan használhatjuk a *System névtér Console osztályát, amely a beolvasás (standard input*, billentyűzet), kiírás (standard output, képernyő műveleteket nyújtja

Output

- Console.Write
- Console.WriteLine

Használat:

- Console.WriteLine(”mit írjon ki {0} {1}…” , parameter1, parameter2, …);



```
int i=5;
Console.WriteLine("Az {0}. művelet eredménye: {1}",i,4*i);
```

Képernyőre: Az 5. művelet eredménye: 20

Példák

```
int i=5;
Console.WriteLine("Az {0}. művelet eredménye: {1}",i,4*i);
Console.WriteLine(" A szám: {0:c}",25);  // pénznem : 25,00 Ft.
Console.WriteLine("A kapott összeg: {0,10:c}",25);
     // 10 karakter széles pénznem formájú kijelzés jobbra igazítva
```

c	pénznembeli (currency) kijelzés
e	tudományos, tízes alapú hatványkijelzés
x	hexadecimális formátum

Általánosan:

{sorszám[, szélesség][:kijelzési_forma]}

Feladatok

Alapadatok

- név (string): Kiss József
- a (int): 25
- b (int): 12

- Kiss József 25 éves
- 12+25=37
- A 12 és a 25 összege 37
- A 25 és a 12 hányadosa: 2,08
- 12 almát vásárolt Kiss József
- A 12 –es szám 25-szöröse 300
- Ma 2014.12.25 van.
- Most 12:25 van.

Beolvasás

char c=Console.Read(); //a c karakter típusú változóba bekér egy karaktert

string s=Console.ReadLine(); //az s string típusú változóba beolvas egy sort amit Enterrel zárunk

Egyéb típusú változóba **át kell konvertálni** a beolvasott adatot:

int j=Convert.ToInt32(Console.ReadLine());

Convert osztály

- ToBoolean
- ToByte
- ToChar
- ToDateTime
- ToDecimal
- ToDouble
- ToInt16
- ToInt32
- ToInt64
- ToSingle
- ToString

Operátorok

- Értékadó
- Matematikai
- Relációs
- Logikai

Értékadás

- int x=10; //x értéke 10
- Típuskényszerítés

```
double d=2.3;
int i=(int) d; // i=2
```

Matematikai műveletek

- + : összeadás
- - : kivonás
- * : szorzás
- / : osztás
- % - maradékképzés

Relációs műveletek

- < kisebb
- > nagyobb
- <= kisebb egyenlő
- >= nagyobb egyenlő
- == egyenlő
- != nem egyenlő
- Eredménye True (1) vagy False (0)

Logikai műveletek

- && - ÉS

a	b	a AND b
1	1	1
1	0	0
0	1	0
0	0	0

- || - VAGY

a	b	a OR b
1	1	1
1	0	1
0	1	1
0	0	0

- ! - tagadás

a	NOT a
1	0
0	1

Rövidítés

- ++x – x értékének növelése 1-gyel
- --x - x értékének csökkentése 1-gyel
- x=x+10 – x értékét növeli 10-zel
- x+=10 – rövid forma, hatékonyabb

Feladatok

1. Kérj be két egész számot, majd írasd ki az összegüket, különbségüket, stb.
2. Kérj be két egész számot, majd írasd ki, hogy igaz-e, hogy az első nagyobb-e, kisebb-e stb. mint a másik
3. Kérj be egy egész számot, majd írasd, hogy igaz-e, hogy a szám a [0;5] intervallumba esik?
4. Kérj be egy egész számot, majd írasd, hogy igaz-e, hogy a szám -5-nél kisebb, vagy 5-nél nagyobb?
5. Kérj be két egész számot két változóba, majd cseréld meg a változók tartalmát!

Feladatok

1. Kérj be egy pénzösszeget, és egy százalékértéket, és növeld a megadott százalékkal! (az eredményt pénznem formátumban írd ki)
2. Kérj be két időértéket (óra:perc), és számold ki a különbséget percben!
3. Kérj be két időértéket (óra:perc), és számold ki a különbséget óra:perc-ben!
4. Kérj be értéket celsius fokban, és számítsd át fahrenheit-be!

Math osztály

Math.Sin(x)	sin(x), ahol az x szög értékét radiánban kell megadni
Math.Cos(x)	cos(x)
Math.Tan(x)	tg(x)
Math.Log(x)	ln(x)
Math.Sqrt(x)	x négyzetgyöke
Math.Abs(x)	x abszolút értéke
Math.Round(x)	kerekítés a matematikai szabályok szerint
Math.Ceiling(x)	felfelé kerekítés
Math.Floor(x)	lefelé kerekítés
Math.Pow(x,y)	hatványozás, x^y
Math.PI	a PI konstans (3.14159265358979323846)

Feladatok

1. Kérd be egy szög értékét fokban, és számítsd át radiánba!
2. Kérd be egy háromszög két oldalát, és a bezárt szögét, majd számold ki a területét!
3. Kérd be egy derékszögű háromszög két befogóját, számold ki az átfogót!
4. Kérd be egy háromszög három oldalát, és számítsd ki a területét!
5. Kérd be egy kör sugarát, számold ki a területét, kerületét!

Véletlenszám generálása

Random osztály Next metódusa:

```
Random r = new Random();
number = r.Next(100);
number = r.Next(10,100);
```

Vezérlési szerkezetek

A program utasításainak sorrendiségét szabályozó konstrukciókat nevezzük.

Ezek lehetnek:

- Szekvencia
- Elágazás
- Ciklus

Szekvencia

- A legegyszerűbb vezérlési szerkezet a szekvencia. Ez tulajdonképpen egymás után megszabott sorrendben végrehajtott utasításokból áll.

Elágazás

- megvizsgálunk egy állítást, és attól függően, hogy igaz vagy hamis, a programnak más-más utasítást kell végrehajtania

```
if (feltétel)
        {
                utasítások
        }
else
        {
                utasítások
        }
```

```
using System;

public class Program
{
	static public void Main()
	{
		int x = 11;

		if(x == 10) // Ha x egyenlő 10 -zel
		{
			Console.WriteLine("x értéke 10");
		}

		if(x != 10)//Ha pedig x nem 10
		{
			Console.WriteLine("x értéke nem 10");
		}
	}
}
```

```
using System;

public class Program
{
	static public void Main()
	{
		int x = 11;

		if(x == 10) // Ha x egyenlő 10 -zel
		{
			Console.WriteLine("x értéke 10");
		}
		else // Ha pedig nem
		{
			Console.WriteLine("x értéke nem 10");
		}
	}
}
```

Több feltétel

- Arra is van lehetőségünk, hogy több feltételt is megvizsgáljunk, ekkor elseif –et használunk:

```
if (feltétel)
        {
                utasítások
        }
else if (feltétel)
        {
                utasítások
        }
else
        {
                utasítások
        }
```

A program az első olyan ágat fogja végrehajtani, amelynek a feltétele teljesül (vagy ha egyik feltétel sem bizonyult igaznak, akkor az else ágat – ha adtunk meg ilyet).

Egy elágazásban pontosan egy darab if, bármennyi elseif és pontosan egy else ág lehet.

Több feltétel 2.

- ha egy változó több lehetséges állapotát akarjuk vizsgálni használhatjuk a switch case utasítást:

```
switch (x)
{       case 10:    utasítások
                    break;
        case 11:    utasítások
                    break;

        default:    utasítások
                    break;
}
```

x értékétől függ melyik ágra kerül a vezérlés

az egyes ágak a kijelölt feladatuk végrehajtása után a break utasítással kilépnek a szerkezetből

megjelenik a default állapot, akkor kerül ide a vezérlés, ha a switch nem tartalmazza a vizsgált változó állapotát

Egyebek…

- A break utasítás csak akkor hagyható el, ha a case után nincs utasítás, ekkor a következő case ág kerül kiértékelésre

```
swith (x)
  {     case 10:
        case 11:

        default:  utasítások
                  break;
  }
```

Egyebek…

- A break utasításon kívül használhatjuk a goto –t is, ekkor átugorhatunk a megadott ágra:

```
swith (x)
   {      case 10:    utasítások
                      goto case 24;

          case 24:    utasítások
                      goto default;

          default:    utasítások
                      break;
   }
```

És még egy elágazás – háromoperandusú művelet

e1 ? e2 : e3

Jelentése: ha e1 igaz, akkor e2, különben e3

Példa:

```
int a = 1;
int b = 2;
int c;
Console.WriteLine((a==b) ? “egyenlő" : “nem egyenlő");
c = (a == b ? 1 : 2);
```

Feladatok

1. Kérj be egy egész számot, és írd ki hogy negatív, vagy pozitív!
2. Kérj be egy egész számot, és írd ki hogy páros, vagy páratlan!
3. Kérd be egy másodfokú egyenlet együtthatóit, és írd ki hány megoldása van az egyenletnek!
4. Kérj be három egész számot, és írd ki a legnagyobbat!
5. Kérj be három egész számot, és írd ki őket növekvő sorrendben!

Feladatok

1. Kérd be egy háromszög három oldalát, és írd ki, hogy megszerkeszthető-e a háromszög?
2. Kérj be egy hónapot sorszámmal, és írd ki melyik hónap!
3. Kérj be egy hónapot sorszámmal, és írd ki melyik negyedévbe esik!
4. Kérj be egy hónapot sorszámmal, és írd ki hány napos a hónap! (szökőévet ne vedd figyelembe)
5. Kérd be egy dolgozat összpontszámát, a tanuló által elért pontszámot, írd ki hány százalékos és milyen érdemjegyű a dolgozat (szövegesen is)!

Iterációk

- Egy adott utasítássorozatot egymás után többször kell végrehajtanunk
- A C# négyféle ciklust biztosít számunkra:
  - Elöltesztelős
  - Hátultesztelős
  - Számlálós
  - Foreach - adathalmaz minden elemére
- Fogalmak
  - Ciklus feltétel – értékétől függ, hogy végrehajtásra kerül a ciklus
  - Ciklusmag – utasítássorozat, amit többször hajtunk végre

Elöltesztelős

```
int i=0;
while (i<10)
  {
      Console.WriteLine(i);
      i++;
  }
```

Ciklusváltozó deklarálása

Ciklus feltétel

Ciklusmag

Mindaddig végrehajtja a ciklusmagot, amíg igaz a feltétel.

Hátultesztelős

```
int i=0;
do
  {
      Console.WriteLine(i);
      i++;
  } while (i<10) ;
```

Ciklusváltozó deklarálása

Ciklusmag

Ciklus feltétel

Mindaddig végrehajtja a ciklusmagot, amíg igaz a feltétel.

Számlálós

```
for (int i=0; i<10; i++)
   {
        Console.WriteLine("Hajrá Szeged!");
   }
```

Honnan?

Meddig?

Hogyan?

Foreach

```
string str = "Hello";
foreach (char ch in str)
   {
         Console.Write(ch);
   }
```

Ciklusváltozó deklarálása

Csak sorszámozott típusra!

Összefoglaló

A ciklus fajtája		A futások száma	Legkevesebb hányszor fut le?
Feltételes	Elöltesztelő	előre nem ismert	lehet, hogy egyszer sem
	Hátultesztelő	előre nem ismert	legalább egyszer
Számlálós		előre ismert	lehet, hogy egyszer sem

Ugró utasítások

- break
- continue
- return
- goto

Break

- Hatására befejeződik a legbelső *while, do, for vagy switch* utasítás végrehajtása. A vezérlés a következő utasításra adódik.
- A break a többirányú elágazás (switch) utasításban is gyakran használt, így kerülhetjük el, hogy a nem kívánt case ágak végrehajtódjanak.

```
int i=1;
while (true) // látszólag végtelen ciklus
   {
         i++;
         if (i==11) break; // Ciklus vége
         Console.WriteLine( i);
}
```

```
int i=1;
while (true && i==11) // látszólag végtelen
ciklus
   {
         i++;
         Console.WriteLine( i);
}
```

Continue

- Hatására a legbelső *while, for, do ciklus utasításokat vezérlő kifejezések* kerülnek kiértékelésre. (A ciklus a következő ciklusmag végrehajtásához készül.)

```
int i=1;
    while(true) // 10 elemű ciklus
        { i++;
          if (i<=10) continue; // következő ciklusmag
          if (i==11) break; // Ciklus vége
          Console.WriteLine( i);
        }
```

Return

- A vezérlés visszatér a függvényből, a kifejezés értéke a visszaadott érték.
- Használata:

```
return ;
return kifejezés;
return (kifejezés);
```

Goto

- A vezérlés arra a pontra adódik, ahol a *címke:* *található*.

Használata:

goto címke;

- A *goto utasításról zárásképpen meg kell jegyezni, hogy a strukturált* programkészítésnek nem feltétlenül része ez az utasítás, így használata sem javasolt.

Példa

```
goto tovább;
Console.WriteLine("Ezt a szöveget sohase írja ki!");
tovább:;
...
int i=1;
switch (i)
{
        case 0: nulla();
                goto case 1;
        case 1:  egy();
                goto default;
        default: valami();
                 break;
}
```

Feladatok

1. Kérj be egy számot 1-10-ig, mindaddig folytasd, amíg jó számot nem ad meg!
2. Kérj be két számot, írd ki az összegüket, majd kérdezze meg a program akar-e még számolni a felhasználó, 1-re tovább számol, 2-re kilép!
3. Írd ki egymás mellé szóközzel elválasztva kettesével a számokat 0-20-ig!
4. Írd ki a négyzetszámokat 1-100-ig!
5. Számold meg mennyi „a” betű van egy felhasználó által megadott karakterláncban!

Feladatok

1. n! – (n faktoriális) – csak 50-nél kisebb számot lehessen megadni!
2. Számkitaláló – a program előállít egy véletlen számot 0 és 100 között, ki kell találni mire gondolt a gép (írja ki, hogy kisebbet vagy nagyobbat kell megadni, majd a végén írja ki, hány lépésben sikerült kitalálni)
3. Írja ki a program, hogy adott szám prímszám-e?
4. Add meg adott szám osztóinak a számát!
5. Számold ki két szám legnagyobb közös osztóját!

Összetett adatszerkezet - Tömb

- Tömb: meghatározott számú, azonos típusú elemek halmaza
- Minden elemre egyértelműen mutat egy index (egész szám)

alma	körte	...	...	banán
0.	1.	2.	3.	4.

Összetett adatszerkezet - Tömb

- Deklarálása:

  típus [] tömbnév = new típus [hossz];

- Pl.: int[] vektor = new int[10];
- Egy tömböt akár a deklaráció pillanatában is feltölthetünk a nekünk megfelelő értékekkel:

  int [] szamok = new szamok[]{ 1,2,3,4,5,6 };

Tömbök bejárása

- for ciklussal:

```
for (int i=0;i<tomb.Lenght;i++)
        {
            Console.WriteLine(tomb[i]);
        };
```

- foreach ciklussal

```
foreach (int tag in tomb)
        {
            Console.WriteLINE(tag);
        };
```

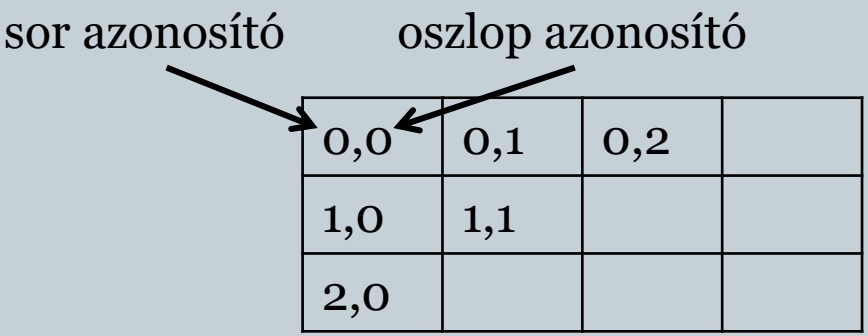
Tulajdonságok, metódusok

- Hossz: tomb.Length
- Rendezés: Array.Sort(tömbnév)
- Átlagszámítás
- Összegzés
- Minimum
- Maximum
- Tartalmazza-e az elemet?

Feladatok

- Olvass be egy tömbbe 10 számot 1 és 100 között.
  - Rendezd a tömböt, és írd ki egymás mellé a számokat
  - Írd ki azokat a számokat, amelyek egy megadott értéknél kisebbek a tömbben
  - Számold ki az összeget, átlagot
  - Írd ki a legnagyobb számot
  - Kérj be egy számot, és írd ki, hogy szerepel-e a tömbben
- Olvass be egy tömbbe keresztneveket
  - Írd ki névsorrendbe a neveket
  - Írd ki csökkenő sorrendbe a neveket
  - Írd ki a leghosszabb nevet
  - Írd ki azokat a neveket, amelyek egy megadott hosszúságúak

Többdimenziós tömbök



- nem egy indexszel hivatkozunk egy elemre, hanem annyival, ahány dimenziós a tömb
- Kétdimenziós tömb:

Pl.: int[,] matrix = new int[3, 4]; //3 sor, 4 oszlop

int[,] tm = new int[2,2] {{1,2},{3,4}}; //kezdőértékkel

Többdimenziós tömbök

- Kezelése : több for ciklus

```
for (int i = 0; i < 4;++i)        //sorok bejárása
    {
            for (int j = 0; j < 3;++j)            //oszlopok bejárása
            {
                    matrix[i, j] = r.Next();
            }
    }
```

1	2	3	4
5	6	7	

Stringek

- Karakterekből áll
- *string* egyes betűire az indexelő operátorral hivatkozhatunk (vagyis minden *string*et kezelhetünk tömbként is)
- Pl.:

```
string szo;
Console.Write(szo[0]); //a szó első karaktere
```

String osztály metódusai

- Length – string hossza
- IndexOf – első előfordulása a megadott karakternek, ha nem tartalmazza -1
- IndexOfAny – első előfordulása a megadott karaktertömb valamelyik elemének
- LastIndex – utolsó előfordulása a megadott karakternek, ha nem tartalmazza -1
- Split – szétszedi a megadott szeparáló karakter szerint egy tömbbe
- Substring – megadott értéktől megadott db karaktert vesz ki
- ToLower – kisbetűssé alakítja a stringet
- ToUpper – nagybetűssé alakítja a stringet
- ToCharArray – karaktertömbbe rakja a string karaktereit

Összetett adatszerkezet - Lista

- Hasonló szerkezetű a tömbhöz
- Nem kell előre megadni a méretét
- Deklarálása:

```
List<int> my_list = new List<int>();
```

Összetett adatszerkezet - Lista

Bejárása 2 féle képpen:

1.
```
foreach (var a in szamok)
    {
        Console.WriteLine(a);
    };
```
2.
```
for (int i = 0; i < szamok.Count; i++)
    {
        Console.WriteLine(szamok[i]);
    };
```

Metódusok

- Elem hozzáadása: my_list.Add(i);
- Elem törlése: my_list.Remove(i);
- Lista teljes tartalmának törlése: my_list.Clear();
- Lista elemeinek száma: my_list.Count;
- Tartalmazza-e a lista az elemet: my_list.Contains(i)
- Hányadik helyen van az elem: my_list.IndexOf(i);

Struktúra adatszerkezet

- Akkor használjuk, ha nem azonos típusú értékeket akarunk együtt kezelni egy tömb típusú adatszerkezetben
- Pl.:
  - Név - string
  - Életkor - integer
  - Neme - bool

név	életkor	Neme
Kiss Géza	34	True
Nagy Ilona	30	False
…		

Megvalósítás 1.

Tömb használatával

```
struct Dolgozok
    {
        public string nev;
        public int kor;
        public bool neme;
    }
static void Main(string[] args)
    {
        Dolgozok [] tabla = new Dolgozok[10]; //felveszünk egy 10 elemű tömböt, amelynek minden eleme Dolgozok típusú
        tabla[0].nev="Kiss Géza"; //tömb 0. indexű elemének feltöltése adatokkal
        tabla[0].kor = 34;
        tabla[0].neme = true;
```

Megvalósítás 1.

Tömb használatával - segédváltozóval

```
 struct Dolgozok
        {
            public string nev;
            public int kor;
            public bool neme;
        }
static void Main(string[] args)
        {
            Dolgozok [] tabla = new Dolgozok[10]; //felveszünk egy 10 elemű
    tömböt, amelynek minden eleme Dolgozok típusú
            Dolgozok egy_dolgozo = new Dolgozok(); //segédváltozó a
    feltöltésre
            egy_dolgozo.nev="Kiss Géza";
            egy_dolgozo.kor=34;
            egy_dolgozo.neme=true;
            tabla[1] = egy_dolgozo;
```

Megvalósítás 2.

Lista használatával - segédváltozóval

```
 struct Dolgozok
        {
            public string nev;
            public int kor;
            public bool neme;
        }
static void Main(string[] args)
        {
            List<Dolgozok> lista = new List<Dolgozok>();
            Dolgozok egy_dolgozo = new Dolgozok();
            egy_dolgozo.nev="Kiss Géza";
            egy_dolgozo.kor=34;
            egy_dolgozo.neme=true;
            lista.Add(egy_dolgozo);
```

Állománykezelés

- A file-okkal kapcsolatos műveletek a System.IO névtérben vannak (using System.IO !!!)
- C# **stream-eket** (adatfolyamokat) használ
- A file-okat **soronként** tudjuk írni, olvasni
- A **fájlmódosítás** alaplépései:
  - A fájl létrehozása vagy megnyitása.
  - Ki- vagy bemenő folyam (stream) hozzárendelése a fájlhoz.
  - A fájl olvasása vagy írása.
  - A folyam, illetve fájl bezárása.

Lehetőségek

- FileStream – folyam megnyitása (létrehozása)
- StreamReader – olvasási folyam
- StreamWriter – írási folyam
  - Paraméterek:
    - Új file neve
    - hozzáfűzzön a file-hoz – true, újat hoz létre – false (felülírja a létezőt)
    - Kódolás : pl.: Encoding.UTF8

Eljárások

- Hosszabb programok írása esetén amennyiben a teljes kódot a Main tartalmazza, a program áttekinthetetlen lesz.
- Javasolt a kód részekre tördelése. Ennek során olyan részeket különítünk el, amelyek önmagában értelmes részfeladatokat látnak el.
- Az ilyen, önálló feladattal és névvel ellátott, elkülönített programrészletet **eljárásnak** nevezzük.
- Az eljárásnak
  - a visszatérési érték típusa kötelezően void,
  - van neve (azonosító),
  - lehetnek paraméterei,
  - van törzse

Példa

```
static void Feladat1()
{
    int a = 10;
    int b = 20;
    Console.WriteLine(a + b);
 }

static void Main(string[] args)
{
     Feladat1();

}
```

Változók használata

- Az eljárásban deklarált változó csak az eljárásban használható
- Ha olyan változót akarunk használni, amit minden eljárás lát, akkor a „kívül” kell deklarálni static jelzővel ellátva

Példa

```
 static int a;                    //globális változó
 static void Feladat1()
{
    int b = 20;                   // helyi változó,  a Feladat1 látja
    Console.WriteLine(a + b);
 }

static void Main(string[] args)
{
    Feladat1();
     a=20;
}
```

Függvények

- A függvény egy olyan eljárás, amely olyan részfeladatot old meg, melynek pontosan egy végeredménye is van – egy érték.
- Amennyiben függvényt akarunk írni, két fontos dolgot kell szem előtt tartanunk:
  - A függvényeknél rögzíteni kell, hogy milyen típusú értéket adnak majd vissza. Ezt a függvény neve előtt kell feltüntetni (a 'void' helyett).
  - A függvények ezek után kötelesek minden esetben egy ilyen típusú értéket vissza is adni! A függvény visszatérési értékét a 'return' kulcsszó után írt kifejezésben kell feltüntetni.

Példa

```
static double Atlag(double a, double b)
        {
            double atl = (a + b) / 2;
            return atl;
        }
static void Main(string[] args)
        {
            Console.WriteLine(Atlag(1, 4));
         }
```