



# C# gyorstalpaló

Készítette:

Major Péter

# Adattípusok

	Típus	Típusnév másképpen (egyenértékű)	Helyigény (bit)	Példa
<b>Logikai változó</b>	bool	Boolean	8 (!)	true, false
<b>Egész szám (*: előjel nélküli)</b>	sbyte, byte*	SByte, Byte	8	-
	short, ushort*	Int16, UInt16	16	-
	int, uint*	Int32, UInt32	32	12, 12u
	long, ulong*	Int64, UInt64	64	12l, 12ul
<b>Lebegőpontos szám</b>	float	Single	32	6.5f
	double	Double	64	6.5
	decimal	Decimal	128	12m
<b>Karakter</b>	char	Char	8	'c'
<b>Szöveg</b>	string	String	változó	"szöveg"

# Változó deklaráció

- Deklaráció, értékkadás:

```
int a;  
a = 5;  
int b = 4, c;  
c = a + b;
```

- minden változónak értékét kell adni, mielőtt felhasználjuk ( pontosabban mielőtt értékét olvasnánk egy művelethez).
- minden utasítást pontosvesszővel zárunk.

# Műveletek

	Operátor	Megjegyzés
Szám típusok	+ , - , *	
	%	maradékképzés
	/	egész típusoknál egész osztás (eredmény), lebegőpontosoknál lebegőpontos
Szöveg	+	összefűzés
Logikai	&&	és
		vagy
	~	negáció (unáris)
Bináris aritmetika	&	és
		vagy
	^	kizáró vagy

Az operátorok összevonhatók értékadással pl.:  $a+=b$ ; ami egyenértékű a következővel:  $a=a+b$ ;

# Blokkok

- A C alapú nyelvekben, így a C#-ban is a kód blokkokban helyezkedik el.
- A blokkokat a { [...blokk tartalma...] } formában jelöljük.
- A blokkok egymásba ágyazhatóak, de nem lapolódhatnak át.

# Ciklusok - for

- Szerkezete:

```
for (int i = 0; i < 10; i++)  
{  
    [...ismétlődő kód...]  
}
```

- Ahol a for kulcsszó utáni zárójeles rész három része:
  - értékadás a ciklusváltozónak (i) – gyakorlatilag bármely utasítás, ami egyszer le fog futni az ismétlés előtt
  - az ismétlés feltétele, addig ismétel amíg igaz – bármely *bool* értékű kifejezés megfelel ide
  - a ciklusváltozó növelése, ciklusonként egyszer hívódik meg ami itt van - tetszőleges utasítás lehet

# Ciklusok - while

- Szerkezete:

```
while ([feltétel])
{
    [...ismétlődő kód...]
}
vagy:
do
{
    [...ismétlődő kód...]
}
while ([feltétel]);
```

- Amíg a feltétel igaz, addig ismétel. A feltétel egy logikai típusú kifejezés.

# Elágazások - if

- Szerkezete:

```
if ([feltétel])
{
    [...kód, ha a feltétel igaz...]
}
else
{
    [...kód, ha a feltétel hamis...]
}
```

- Az else elhagyható.
- Ha csak egy utasításból állna az igaz vagy hamis ág, akkor nem kell blokkot használni.

# Elágazások - switch

- Egy változó értékétől függően, más-más kódrész fut le.
- A default ág akkor fut le, ha a változó értéke egyik megadott case-el sem egyezik meg.
- A default ág elhagyható.

```
switch ( [változó] )
{
    case [érték1]:
        [...kód...]
        break;
    case [érték2]:
    case [érték3]:
        [...kód...]
        break;
    default:
        [...kód...]
        break;
}
```

# Tömbök

- Egyező típusú adatok sokaságának tárolására szolgál.
- Használata (10 elemű tömbbel):

```
int [] tomb;  
tomb = new int [10];  
  
tomb [0] = 4;  
tomb [1] = 4 + tomb [0];
```

- Használat előtt helyet kell foglalni a tömb számára, ilyenkor megadjuk a tömb méretét.
- Az elemek a [] operátorral érhetők el.
- A tömb mérete a `tomb.Length` jellemzővel kapható meg.
- Számozásuk mindig 0-tól a `tomb.Length-1`-ig tart.

# Többdimenziós tömbök

- Használat:

```
int[,] tomb;  
tomb = new int[10,15];
```

```
tomb[0,2] = 4;  
tomb[1,1] = 4 + tomb[0,2];
```

- A különböző dimenziókra vonatkozó méreteket ill. pozíciókat vesszővel választjuk el.
- A **Length** mező ilyenkor az összes elem száma.
- A dimenziónkénti elemszám az *int GetLength( int dimension)* metódussal kapható meg.

# Matematikai műveletek

- A **System.Math** osztály statikus függvényeivel:
  - Trigonometrikus függvények:
    - **Sin, Cos, Tan, Asin, Acos, Atan**
  - Hiperbolikus függvények:
    - **Sinh, Cosh, Tanh, Asinh, Acosh, Atanh**
  - Exponenciális, logaritmikus, hatványozó:
    - **Exp, Pow, Sqrt, Log** (természetes v. tetszőleges alapú), **Log10**
  - Kerekítés:
    - **Round**
    - **Truncate** (egészrész)
  - Abszolút érték: **Abs**
  - Szignum függvény: **Sign**
  - Konstansok: **E, Pi**

# Szövegek kezelése

- Szöveg konstansok, összefűzés, adott karakter kiolvasása:

```
string szoveg = "hello";
szoveg += " world";
char c = szoveg[0]; //c=='h'
```

- A *stringek* kezelésére szolgáló függvények a *string* osztály metódusai, tehát tetszőleges *string* változó után pontot téve elérhetők.
- Ezek a metódusok mindig új szöveg objektumot hoznak létre, ezért teljesítmény szempontjából érdemes lehet a *StringBuilder* osztály használata.

# Stringkezelő függvények

- *int IndexOf( string value )*: szöveg első előfordulásának meghatározása
- *int LastIndexOf( string value )*: szöveg utolsó előfordulásának meghatározása
- *string Insert( int startIndex, string value )*: szöveg beszúrása adott pozícióba
- *int Length { get; }*: szöveg hossza
- *string PadLeft( int totalWidth, char paddingChar )*: kiegészítés adott szélességre adott karakterrel balról
- *string PadRight( int totalWidth, char paddingChar )*: kiegészítés adott szélességre adott karakterrel jobbról
- *string Remove( int startIndex, int count )*: adott pozíciótól adott számú karakter eltávolítása
- *string Replace( string oldValue, string newValue )*: keresés és csere
- *string[] Split( string[] separator, StringSplitOptions options )*: szöveg szétvágása adott határoló stringel

# Stringkezelő függvények

- *string[] Split( char[] separator, StringSplitOptions options )*: szöveg szétvágása adott határoló karakterrel
- *bool StartsWith( string value )*: visszaadja, hogy ezzel a karakterláncjal kezdődik-e a szöveg
- *string Substring( int startIndex, int length )*: adott kezdőpozíciótól adott hosszúságú szövegrészt adja vissza
- *string ToLower()*: átalakítás kisbetűssé
- *string ToUpper()*: átalakítás nagybetűssé
- *string Trim( params char[] trimChars )*: adott karakterek levágása a szöveg elejéről és végéről
- *string TrimEnd( params char[] trimChars )*: adott karakterek levágása a szöveg végéről
- *string TrimStart( params char[] trimChars )*: adott karakterek levágása a szöveg elejéről
- *bool EndsWith( string value )*: visszaadja, hogy ezzel a karakterláncjal végződik-e a szöveg