C# alapjai

Környezete

- Maga a .NET platform a Microsoft, a Hewlett Packard, az Intel és mások közreműködésével megfogalmazott CLI (Common Language Infrastructure) egy implementációja. A CLI egy szabályrendszer, amely maga is több részre oszlik:
  - A **CTS** (Common Type System) az adatok kezelését, a memóriában való megjelenést, az egymással való interakciót stb. írja le.
  - A **CLS** (Common Language Specification) a CLI kompatibilis nyelvekkel kapcsolatos elvárásokat tartalmazza. ·
  - A **VES** (Virtual Execution System) a futási környezetet specifikálja, nevezik CLR - nek (Common Language Runtime) is.

A valóságban nincs .NET virtuális gép, helyette ún. felügyelt (vagy managed) kódot használ, vagyis a program teljes mértékben natív módon, közvetlenül a processzoron fut, mellette pedig ott a keretrendszer, amely felelős pl. a memóriafoglalásért vagy a kivételek kezeléséért. A .NET nem programozási nyelv, hanem környezet. Gyakorlatilag bármelyik programozási nyelvnek lehet .NET implementációja. Jelenleg kb. 50 nyelvnek létezik hivatalosan .NET megfelelője

A .NET (akárcsak a Java) más úton jár, a fordító először egy köztes nyelvre (Intermediate Language) fordítja le a forráskódot. Ez a nyelv a .NET világában az MSIL, illetve a szabványosítás után a CIL (MICROSOFT/CommonIL) – különbség csak az elnevezésben van.

C# tisztán **objektumorientált**, típusbiztos, általános felhasználású nyelv

C# szintaktikája

- Az egyes utasítások végén pontosvessző - ; - áll.
- A kis- és nagybetűk különböző jelentőséggel bírnak, azaz a “program” és “Program” azonosítók különböznek. Ha a fenti kódban Console.WriteLine helyett console.writeline –t írnánk, akkor a program nem fordulna le.
- A program egységeit (osztályok, metódusok stb.) ún. blokkokkal jelöljük ki, kapcsos zárójelek ({ és }) segítségével.

Kulcsszavak

- Olyan szavak, mellyel speciális jelentőséggel bírnak, így másra a programban NEM lehet felhasználni

MEGJEGYZÉSBE lehet írni

- Egysoros megjegyzés: // jel után

Pl. **// Ez egy egysoros komment**

- Többsoros megjegyzés: /* és */ jel közé

Pl. /* **Ez egy**
többsoros
komment */

Névterek

- Tulajdonképpen egy virtuális doboz, amelyben a logikailag összefüggő osztályok, metódusok stb. vannak
- Más megfogalmazásban: A névtér (namespace) az a logikai egység, amiben az azonosítónknak egyedinek kell lennie

- Nyilván könnyebb megtalálni az adatbázis-kezeléshez szükséges osztályokat, ha valamilyen kifejező nevű névtérben vannak (pl. System.Data).
- Használatba venni a using kulcsszóval kell

Pl. using System;

- Lehet saját névteret létrehozni, de nem kötelező, ebben az esetben az ún. név nélküli névtér része lesz az adott kód.

Névtér szerkezete:

```
namespace új_névtérnév
{
        class új_osztálynév
        {
                Típusdefiníció;
                Függvénydefiníció;
        }
...
}
```

Változók

- Olyan tárolók, ahová az adatainkat ideiglenesen eltároljuk
- A változók a memória egy (vagy több) cellájára hivatkozó leírók. Egy változót a következő módon hozhatunk létre C# nyelven:
  - **Típus változónév;**
- A változónév első karaktere csak betű vagy alulvonás jel (_) lehet, a többi karakter szám is.
- Lehetőleg kerüljük az ékezetes karakterek használatát (bár technikailag nem okoz problémát, nem akadályozza a fordítást)!
- Konvenció szerint a változónevek kisbetűvel kezdődnek. Amennyiben a változónév több szóból áll, akkor célszerű azokat a szóhatárnál nagybetűvel “elválasztani” (pl. pirosAlma, vanSapkaRajta stb.).

Deklaráció és Definíció

- Egy változó (illetve lényegében minden objektum) életciklusában megkülönböztetünk deklarációt és definíciót.
- A deklarációnak tartalmaznia kell a típust és azonosítót
- a definícióban pedig megadjuk az objektum értékét **Értelemszerűen a deklaráció és a definíció egyszerre is megtörténhet.!!!**

```
int x; // deklaráció
x = 10; // definíció
int y = 11; // deklaráció és definíció
```

Típusok

- A típus határozza meg, hogy egy változó milyen értékeket tartalmazhat, illetve mekkora helyet foglal a memóriában
- Ha nem tudjuk meghatározni a típust, a fordítóra bízhatjuk, csak akkor a var kulcsszót kell a változó elé tenni

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

Érték és Referenciatípusok

- Attól függ, hogy hol tárolja a program
- Értéktípus-→ stack(verem/LIFO)
- Referenciatípus→heap (halom)

Programozói szinten számunkra nincs jelentősége, hogy hol tárolódik a változó, és miként.

Felsorolás típus

- Olyan adatszerkezet, amely meghatározott értékek névvel ellátott halmazát képviseli.
- Felsorolt típust az enum kulcsszó segítségével deklarálhatunk
- Példa: Enum allatok{medve,delfin,capa};

Konstans

- A const típusmódosító kulcsszó segítségével egy objektumot konstanssá, megváltoztathatatlanná tehetünk.
- A konstansoknak egyetlenegyszer adhatunk (és ekkor kötelező is adnunk) értéket, mégpedig a deklarációnál.

Példák:

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

```
Console.WriteLine("Adjon meg egy számot: ");
const int x = int.Parse(Console.ReadLine());
```

Kifejezések,Operátorok, Operandusok

- Az utasítások kifejezésekből állnak, a kifejezések pedig operátorokból és operandusokból, illetve ezek kombinációjából jönnek létre
- Példa:

Szam = a – b **a, b operandus** **- operátor**

Precedencia szabály

- Több operátor együttes előfordulása esetén fellépő „kiértékelési” szabály (a sorrend fentről lefele)

zárójel
előjel (negálás)
szorzás, osztás (maradékos/maradék nélküli)
összeadás, kivonás
bit eltoló operátorok (>>,<<)
<,>,<=,>=
egyenlő, nem egyenlő
Logikai és (AND)
Logikai kizáró vagy (XOR)
Logikai vagy (OR)

Példák operátorokra

- Értékadó pl. x=10;
- Matematikai *,% (maradékos), / (maradék nélküli)
- -,+
- Relációs

x > y	x nagyobb, mint y
x >= y	x nagyobb vagy egyenlő, mint y
x < y	x kisebb, mint y
x <= y	x kisebb vagy egyenlő, mint y
x == y	x egyenlő y-nal
x != y	x nem egyenlő y-nal

- Logikai && ÉS operátor „igazságtáblája”
- II VAGY operátor „igazságtáblája”

A	B	Eredmény
hamis	hamis	hamis
hamis	igaz	hamis
igaz	hamis	hamis
igaz	igaz	igaz

A	B	Eredmény
hamis	hamis	hamis
hamis	igaz	igaz
igaz	hamis	igaz
igaz	igaz	igaz

- Logikai ! TAGADÁS operátora
- (feltételes) „igazságtáblája”

EGY operandusú!!!!

A	Eredmény
hamis	igaz
igaz	hamis

Mit jelent a lusta kiértékelés? (lásd órai példa)

- Feltételes operátor az egyetlen 3 operandusú

Feltétel ? Igaz-ág: Hamis-ág; (If-Then-Else kiváltása)

Rövidítési lehetőségek

- Lehetőségünk van értékadásnál különböző rövidebb írásmódot használni
- Az összes aritmetikai operátornak létezik rövid formája

Pl.
X=X+20 x+=20
X=X+1 prefixes ++X; postfixes X++ (előbb átmeneti vált.

VEZÉRLÉSI SZERKEZETEK
1_Szekvencia

- Azt jelenti, hogy sorban hajtja végre a kiadott parancsokat

VEZÉRLÉSI SZERKEZETEK
2_Feltételes utasítások

- Kétirányú elagázás_ szintaxisa:

```
if (feltétel)
        { igaz állítás esetén végrehajtandó utasítások}
else
        { hamis állítás esetén végrehajtandó utasítások}
```

Megjegyzés: Nem kötelező az else ág (szervezés?!)
egymásba is ágyazható (else if)
összetett kifejezés is írható feltételként
Egy elágazásban pontosan egy darab if, bármennyi else-if és pontosan egy else ág lehet.

VEZÉRLÉSI SZERKEZETEK
3_Switch_Case szerkezetek

- Többirányú elágazásnak is nevezik
- Szintaxisa:

```
switch (x)
{
        case 10:
         Console.WriteLine("x értéke 10");                    break;
        case 11:
Console.WriteLine("x értéke 11");                    break;
        default:
                        Console.WriteLine(„nem tudom");
                        break;
}
```

VEZÉRLÉSI SZERKEZETEK
4_Ciklusok

- Ismétlődő tevékenységek végrehajtására szolgál.
- Fontos jellemzők: Hányszor fut le, lehet-e végtelen ciklus vagy üres ciklus, hova kerül a ciklus feltétele (ha van).
- Fajtái:
  - Elöltesztelő
  - Hátultesztelő
  - Előírt lépésszámú
  - Iterációs/Foreach
- Részei: ciklusfej, ciklusmag, ciklusvég

Elöltesztelő ciklus

Szintaxisa:
while (feltétel)
{
Utasítások;
}

Működése: a ciklusmag végrehajtása előtt ellenőrzi a ciklusfeltételt, ha igaz a feltétel, a ciklusmagban elhelyezkedő utasítások végrehajtódnak
Megjegyzés: előfordulhat az is, hogy a ciklustörzs egyszer sem fut le

Hátultesztelő ciklus

Szintaxis:
do
{
Utasítások;
} while (feltétel);

Működése: ciklusmag végrehajtása után ellenőrzi a ciklusfeltételt
Megjegyzés: a ciklustörzs legalább egyszer lefut

Előírt lépésszámú ciklus

Szintaxisa:

```
for  (cik_vált_típus cik_vált_kezdő; meddig;lépésköz)
 {
        utasítások;
 }
```

Példa:

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

for  (int i = 2; i < 10; i+=2)
 {
Console.WriteLine(i);
}
```