Kezdő C# programozás szakkör jegyzet

Szkupien Péter

2019. október*

1. Jegyzet

Ez a jegyzet a Szent István Gimnázium 2019/2020-as kezdő C# szakköréhez készült oktatási segédletként, nyelvezete és precizitása a szakkör hetedikes hallgatóságához igazodik. Ha elírást vagy hibát találsz benne, megköszönöm, ha jelzed a peti.szkupien@gmail.com email címen. Ha valamit nem értenél akár a jegyzetből, akár egy otthoni próbálkozás eredményéből, nyugodtan írj egy emailt, és szívesen válaszolok.

2. Program, programozás madártávlatból

Ha nagyon nagy vonalakban tekintünk a számítógépre és a rajta futó programokra, lényegében ugyanúgy működnek, mint bármilyen más kommunikáció is: vagy a felhasználó „mond valamit" a számítógépnek, vagy a számítógép „mond valamit" a felhasználónak. A programjaink, amiket írni fogunk, ezt a két dolgot fogják összekötni: lényegében azt a logikát fogjuk megvalósítani, ami összeköti a felhasználó és a számítógép tevékenységeit. (Pl.: a felhasználó beírja, hogy 2+2, a program kiírja, hogy 4. A kettő közti logika az, ami a programunk lényegét adja: értelmezni, hogy kettő számot és egy műveleti jelet kaptunk, rájönni, hogy az milyen művelet, elvégezni a műveletet, majd kiírni az eredményt. Ugye, hogy nem is olyan egyszerű?)

3. Konzol

A fent leírt kommunikációt egy egyszerű konzolon keresztül fogjuk megvalósítani. Természetesen vannak csili-vili felületek is, ahol szép rózsaszín gombokat gyárthatunk, de ezek ugyan szépek, de cserébe talán el is terelik a figyelmet a lényegről. A konzol, amit használni fogunk, egy nagyon egyszerű fekete hátterű ablak, amire fehér karakterek írhatók. Vagy a programunk ír ki rá valamit a felhasználónak, vagy a felhasználó ír be valamit a programnak. Ennyire egyszerű.

*Ez a jegyzet eredetileg Wordben készült, 2022. szeptemberében alakítottam át LaTeX-alapúvá.

4. Visual Studio, C#

A Microsoft **Visual Studio** nevű fejlesztőkörnyezetében fogunk programozni, C# nyelven. A Visual Studio egy rendkívül összetett fejlesztőeszköz, aminek mi értelemszerűen csak néhány nagyon alapvető funkcióját fogjuk használni. A **C#** (*ejtsd: szí sárp*) egy magasszintű programozási nyelv, vagyis olvasható kódot írhatunk benne, viszonylag könnyen.

A Visual Studio az iskolai gépekre fel van telepítve, de otthon is elérhető ingyenesen: a Visual Studio Community 2019 letölthető a visualstudio.com oldalról, a használatához csak be kell jelentkezni egy tetszőleges Microsoft-fiókba (ez lehet akár az iskolai is, akár egy „privát" is).

5. Projekt

A Visual Studio-ban írt programjaink projektekbe rendeződnek, vagyis, ha új programot szeretnénk írni, új projektet kell létrehoznunk (*természetesen egy projektben több fájl is lehet, de ennek a hogyanjának a megértéséhez olyan ismeretekre lenne szükség, amelyeknek egyelőre nem vagyunk birtokában, így fogadjuk el az ökölszabályt, hogy mindig új projektet hozunk létre*).

Új projektet a New project menüponttal hozhatunk létre (ennek a menüpontnak a helye függ a konkrét Visual Studio-verziótól, de az indításkor megjelenő képernyőn mindig könnyen megtalálható). Válasszuk a C# nyelvű Console Application (.NET Framework) projekttípust. Ezután megadható a projekt neve, valamint a mappa, ahová menteni szeretnénk (általánosságban elmondható, hogy érdemes az elérési utakban és nevekben tartózkodni az ékezetektől, szóközöktől, különleges karakterektől stb.)

A kiválasztott helyre létrejött projektünk sok fájlt tartalmaz, amelyekre általában nekünk nincs szükségünk, de a Visual Studionak fontosok, így természetesen nem szabad kitörölni őket. Ha a programkódunkat keressük, a projektmappában a projekt neve/Program.cs fájlban találjuk, a futtatható .exe fájl pedig a projekt neve/bin/Debug helyen jön létre (az első fordításkor).

Ha létrejött a projektünk, megismerkedhetünk a Visual Studio felépítésével. A funkciók nagy részére nincs szükségünk, a legfontosabbak:

- **Solution Explorer**, általában az ablak jobb szélén: itt találjuk a projektünk felépítését, innen tudjuk megnyitni a Program.cs fájlt, amelyikbe a programkódot írjuk, ha véletlenül bezárnánk. Ha nem látjuk a Solution Explorert, a View/Solution Explorer menüponttal megjeleníthetjük.
- **Programkód**, középen: itt szerkeszthetjük a programunk kódját.
- **Error List**, általában az ablak alján: itt olvashatjuk a programunkban lévő (szintaktikai) hibákat, amennyiben vannak ilyenek. Ezek angol nyelvű hibaüzenetek, de nagyon hasznos, ha valaki tudja őket értelmezni. Ha nem látjuk az Error Listet, a View/Error List menüponttal megjeleníthetjük.

A programunkat az F5 gombbal, vagy az eszköztáron található zöld színű nyíllal jelzett Start gombbal indíthatjuk el.

6. Hello World!

A létrejött Program.cs fájlunk egyáltalán nem üres, elsőre sok furcsa karaktersorozatot találunk benne. A jó hír, hogy ezekkel többnyire egyelőre semmi dolgunk nem lesz, korán vagyunk még ahhoz, hogy megértsük, mi mit jelent. Alapértelmezetten ezt kell látnunk:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
        class Program
        {
                static void Main(string[]
                    args)
                {
                }
        }
}
```

Programkód 1. Hello World!

Amit talán érdemes megjegyeznünk, hogy a `static void Main` kezdetű sor a programunk *belépési pontja*, vagyis itt fog elkezdeni igazából futni. Az ezt a sort követű `{ }`-ba (kapcsoszárójelpár, blokk) fogjuk írni egyelőre a teljes programkódunkat. Ennek megfelelően a további példa programkódoknál nem írjuk le mindig ezt a teljes „körítést", csak a `Main`-en belüli részre szorítkozunk.

Hagyományosan minden programozást oktató könyv/kurzus egy olyan programmal kezdődik, ami kiírja, hogy "Hello World!", így mi is ezzel kezdünk. A korábban már említett konzolra a `Console.WriteLine("HelloWorld!");` sorral tudjuk ezt kiírni.

Figyeljük meg ezt az egyszerű sort! A `Console` az a *valami*, amit használni szeretnénk, a `WriteLine` az a *valami*, amit csinálni szeretnénk a konzollal, a kettő között pedig egy pont áll. Az elnevezések angolok, nagybetűvel kezdődnek, a többszavas nevekben pedig a szóhatárt szintén nagybetűvel jelöljük. A `WriteLine`-t egy zárójelpár követi, amiben idézőjelek között áll a kiírandó szöveg. A sort egy pontosvessző zárja.

Nagyon nem precíz, de egyelőre praktikus ökölszabályként elmondható, hogy azokat a nevesített utasításokat, amik *valamivel csinálnak valamit* zárójelpár követi, amiben (amennyiben vannak) megadhatjuk azokat a részleteket, amik pontosítják, hogy mi fog történni. Ebben az esetben az utasítás, ami *valamivel csinál valamit* a `WriteLine`, hiszen kiír valamit a konzolra. A zárójelpárban pedig megadhatjuk, hogy mit írjon ki, ez lényegében pontosítja a működést. (*Szakkifejezéssel élve a `WriteLine`-t függvénynek vagy metódusnak, a zárójelpárban lévő dolgokat pedig paramétereknek nevezzük.*) A kiírandó szöveget idézőjelek közé írjuk, ezzel jelezzük a számítógépnek, hogy ez egy sima szöveg, nem pedig egy utasítás. Minden utasítás sor végére pontosvesszőt teszünk. (Alt Gr+?)

Ha nem azt szeretnénk kiírni, hogy HelloWorld!, csak egy üres sort, a zárójelpárt üresen hagyhatjuk, de akkor sem hagyhatjuk el.

6.1. Nem működik a program?!

Ha a fenti Hello Worl! programot lefuttatjuk, valószínűleg nem fogjuk látni a hőn áhított eredményt, legfeljebb egy pillanatra felugró, majd rögtön el is tűnő fekete ablakot. Ennek az az oka, hogy a programunk sorra hajtja végre az utasításokat, majd, ha a végére ért, egyszerűen leáll. Ez történik itt is, olyan gyorsan írja ki ezt az egy sort, majd áll is le rögtön, hogy mi azt észre sem vesszük. Ha szeretnénk megbizonyosodni róla, hogy valóban kiírtuk, amit akartunk, tegyük a programunk legvégére a következő sort:

```
Console.ReadKey();
```

Programkód 2. Várakozás billentyűleütésre

Ennek hatására a programunk a végén várni fog tőlünk egy billentyűleütést, és csak utána fog bezáródni. Azt, hogy ez pontosan mit jelent, a későbbiekben tárgyaljuk.

6.2. Megjegyzések

A programkódunkba írhatunk megjegyzéseket, amiket a számítógép figyelmen kívül hagy, de magunknak hasznosak lehetnek, átláthatóbbá tehetik a programkódunkat. Egysoros megjegyzést `//` után írhatunk, több sorost pedig a `/* ... */` közé.

```
// ez egy megjegyzes, barmit irhatunk ide,
    nem szamit
/* ez egy tobb soros megjegyzes
egeszen addig, amig
le nem zarjuk */
```

Programkód 3. Megjegyzések

7. Változók

A programjaink persze nem sokat érnének, ha mindent előre belejük kellene írnunk. Adja magát az igény, hogy valahogy el tudjunk tárolni értékeket, és azokat utána (újra) fel tudjuk használni. Például, ha a programunk egyszerűen megszorozza kettővel a kapott számokat, akkor is először el kell tárolnunk a beírt számot, hogy utána meg tudjuk szorozni kettővel.

Képzeljük el ezt úgy, mintha lennének dobozaink, amikbe be tudunk tenni dolgokat, később pedig meg tudjuk nézni, mi van bennük, és le is tudjuk cserélni a tartalmukat. A dobozaink viszont különböző formájúak, és mindegyikbe csak olyan formájú dolgot tehetünk, amilyen a doboz.

A dobozainkat használni is szeretnénk később, így megkülönböztethetők kellenek, hogy legyenek. Mindegyiknek kell, hogy legyen tehát egy egyedi neve, ami beazonosítja.

A fenti mondatok lényegében meg is határozták, mit értünk változó alatt: egy típussal és egyedi névvel rendelkező valami, ami típusának megfelelő értéket tud tárolni, amit ki tudunk olvasni és meg tudunk változtatni. (Innen ered a változó elnevezés: egy olyan logikai egység („doboz”), aminek az idő múlásával változhat az értéke.)

A változó típusa lényegében bármi lehet, a teljesség igénye nélkül a legelemibbek:

- **egész szám** (`int`)
- **tört szám** (`float`): programozásban az angol nyelvhez hasonlóan tizedespontot használunk, nem tizedesvesszőt
- **karakter** (`char`): ez egyetlen karaktert jelent, ami persze nem csak betű lehet, hanem számjegy, írásjel, bármi. A karaktereket szimpla idézőjelbe tesszük, pl.: 'a'
- **szöveg** (`string`): bármilyen hosszú lehet a szövegünk (vagyis akár üres is), amit (dupla) idézőjelbe teszünk, pl.: "szöveg", "" (utóbbi az üres string)
- **logikai érték** (`bool`): igaz (`true`) vagy hamis (`false`)

Most, hogy tudjuk, mit jelent a változó, és milyen típusai lehetnek, már csak az a kérdés, hogyan használjuk őket a programkódunkban. Létrehozhatunk egy változót anélkül, hogy megadnánk a kezdeti értékét (természetesen valamilyen alapértelmezett értéke ekkor is lesz, csak azt nem mi adjuk meg explicit – szám esetén 0, logikai érték esetén a hamis az alapértelmezett érték, de mindig adjunk értéket a változóinknak, mielőtt használnánk őket!) és úgy is, hogy rögtön értéket is adunk neki (előbbit deklarációnak, utóbbit definíciónak hívjuk). Változót a típus és a változónév megadásával hozhatunk létre, a pontosvessző természetesen kell a sor végére.

```
int szam = 5;     // szam nevu int tipusu
    valtozo, 5 ertekkel
string szoveg;    // szoveg nevu string
    tipusu valtozo
szoveg = "szia"; // kesobb adtunk erteket
    neki
```

Programkód 4. Változók

7.1. Értékadás és egyenlőségjel

Fontos tisztázni, mit is jelent a programjainkban az egyenlőségjel. Nézzük például a következő kódot!

```
int szam = 5; // ez utan a sor utan szam
    erteke 5
szam = 10;    // ez utan a sor utan szam
    erteke 10
```

Programkód 5. Értékadás

Az első sorban létrehoztunk egy `szam` nevű, `int` típusú változót, aminek 5-öt adtunk értékül. A második sor viszont már érdekesebb: az egyenlőségjel bal oldalán egy 5 értékű változó áll, a jobb oldalon pedig a 10. Jól jegyezzük meg, ez nem azt jelenti, hogy ez a két dolog egyenlő ($5 \neq 10$), hanem hogy a `szam` nevű változónak értékül adjuk a 10-et. Az egyenlőségjel tehát programozásban az értékadás jele, nem pedig az egyenlőségvizsgálaté (arról, hogy utóbbinak akkor mi a jele, hamarosan).

Ne felejtsük el, hogy a változóneveknek egyedieknek kell lenniük! A fenti példák külön-külön értelmezendők, ezt az öt sort egymás után leírva hibás programot kapnánk, hiszen kétszer hozunk létre `szam` nevű változót.

8. Beolvasás

Azt már tudjuk, hogyan tudunk kiírni a konzolra, azt viszont még nem, hogy hogyan tudunk onnan beolvasni a felhasználótól. Ezt nem véletlenül a változók után tárgyaljuk: a beolvasásnak akkor van értelme, ha el is tudjuk tárolni a beolvasott érétket, ezt pedig egy változóval tudjuk megtenni.

```
string szoveg = Console.ReadLine();
```

Programkód 6. Beolvasás

Így kell létrehozni egy `szoveg` nevű, `string` típusú változót, és beolvasni bele egy sort a konzolról. A `ReadLine()` ugyanúgy metódus, mint a `WriteLine()` volt, vagyis ennek a végére is zárójelpárt kell tennünk. A fenti példában tehát a `Console.ReadLine()` beolvas egy sort a felhasználótól, majd ezt értékül adjuk a frissen létrehozott `szoveg` nevű változónak.

A `Console.ReadLine()` egy sort olvas be a felhasználótól, a sorokat pedig enterrel zárjuk. Vagyis egy `ReadLine`-os programsor mindig vár egy entert, másképp fogalmazva a programunk egészen addig nem megy tovább, amíg a felhasználó entert nem üt.

Most már van elég tudásunk ahhoz, hogy megértsük, miért írtunk a programjaink végére `Console.ReadKey();`-t. A `Console.ReadKey()` ugyanúgy vár egy billentyűleütést, ahogy a `Console.ReadLine()` egy entert a sor végén. A különbség csak annyi, hogy míg a `ReadLine` által beolvasott sort eltároljuk egy változóba, a `ReadKey`-t nem erre használtuk, hanem csak arra, hogy várjon a programunk egy billentyűleütésig. (Egyébként a `ReadKey`-jel beolvasott valamit is el lehetne tárolni, de ennek a típusa és a működése meghaladja jelenlegi tudásunkat.)

Az előző példában szöveget olvastunk be a konzolról, amit eltároltunk egy szöveg típusú változóba. Ha azonban számot szeretnénk beolvasni, egy kicsit nehezebb dolgunk van. A következő sor ugyanis NEM működik, hiába írná be a felhasználó a konzolra, hogy 123.

```
int szam = Console.ReadLine(); //ez NEM
    mukodik
```

Programkód 7. Hibás beolvasás

A problémát az okozza, hogy a `ReadLine()` egy szöveget (`string`-et) olvas be a konzolról, az `int` típusú változóba pedig nem tehetünk `string`-et (emlékezzünk vissza: a kör alakú dobozba nem passzol a négyzet). A beolvasott szövegből tehát valahogyan számot kell „csinálnunk”.

```
int szam = int.Parse(Console.ReadLine());
```

Programkód 8. Egész szám beolvasása

Erre szolgál az `int.Parse()`, ami a „hasába betett” szövegből `int`-et csinál (szebben mondva: az *argumentumként* kapott `string`-et `int`-té konvertálja). Ezt ugyanígy megtehetjük törtek esetében is, csak `int` helyett `float`-ot kell írnunk:

```
float tort = float.Parse(Console.ReadKey());
```

Programkód 9. Törtszám beolvasása

Azt már tudjuk, hogy programkódban a törtekben tizedespontot használunk, de beolvasásnál ez egy kicsit bonyolultabb. Az, hogy vesszőt vagy pontot kell-e használnunk, az operációs rendszerünk területi beállításaitól függ (magyar Windows esetén vessző, angol esetén pont).

9. Változók értékének kiírása

Azt már láttuk, hogyan írhatunk ki egy általunk beírt szöveget a konzolra, ez azonban önmagában nem túl hasznos. A programunknak pont az a lényege, hogy reagálni tud a felhasználó bemeneteire, vagyis nyilvánvalóan nem tudjuk (és nem is akarjuk) előre beírni az összes szöveget, amit majd valaha ki szeretnénk írni. Adja magát tehát az igény, hogy a konzolon megjelenő szövegbe változók értékeit is belefűzzük. Ezt többféle módon is megtehetjük, a példákban beolvasunk két számot, és összeadjuk őket, majd kiírjuk az eredményt „a és b összege c” alakban.

9.1. Egy nem javasolt módszer

Az elsőre legegyszerűbbnek tűnő módszer többnyire ugyan működik, de nagyon nem elegáns és könnyű elhibázni. Az alapja, hogy a szövegeket `+` jellel össze tudjuk adni, így a `Console.WriteLine()` zárójelpárjában az idézőjelek közt lévő szövegek és a változónevek közé egyszerűen `+` jelet teszünk.

```
int a = int.Parse(Console.ReadLine()); // 1
int b = int.Parse(Console.ReadLine()); // 2
int ossz = a + b;                      // 3
Console.WriteLine(a + " es " + b + "
    osszege " + ossz);
// 1 es 2 osszege 3
```

Programkód 10. Változók értékének kiírása – nem elegánsan

9.1.1. A probléma

A fenti módszerrel van azonban egy probléma, amibe könnyen beleszaladhatunk. A `+` jel nem mindig azt csinálja, amit várnánk tőle. A fenti kódot egy kicsit másképp írva elsőre meglepő eredményt kapunk.

```
int a = int.Parse(Console.ReadLine()); // 1
int b = int.Parse(Console.ReadLine()); // 2
Console.WriteLine(a + " es " + b + "
    osszege " + a+b);
// 1 es 2 osszege 12
```

Programkód 11. Változók értékének kiírása – hibásan

A fenti kód ugyan lefut, de elsőre azt hihetnénk, elromlott a számítógép, hiszen a konzolon a következő jelenik meg: `1 es 2 osszege 12`

Ennek az oka, hogy a `Console.WriteLine()` zárójelpárjában a számítógép balról jobbra haladva elkezdi elvégezni a műveleteket, de mivel a `+` jel egyik oldalán szám, a másik oldalán pedig szöveg áll, azokat szövegként fűzi össze, nem pedig számként adja össze. Persze ha az `(a+b)`-t zárójelbe tesszük, már működik, de ezt nagyon könnyű elfelejteni, és a programunk még csak nem is fog nekünk szólni, hogy elrontottunk valamit, egyszerűen csak (számunkra) butaságot fog kiírni.

Kezdőként elég nehéz mindig átlátni, hogy az ilyen műveletek esetén pontosan mi történik, és egyéb mélyebb okai is vannak, hogy nem célszerű így összefűzni szövegeket. Röviden tehát annyit jegyezzük meg, hogy a fenti módszert NE használjuk.

9.2. Az elegáns módszer

```
int a = int.Parse(Console.ReadLine()); // 1
int b = int.Parse(Console.ReadLine()); // 2
int ossz = a + b;                      // 3
Console.WriteLine("{0} es {1} osszege {2}",
    a, b, ossz);
//1 es 2 osszege 3
```

Programkód 12. Változók értékének kiírása – elegánsan

A `Console.WriteLine()` zárójelpárjába, az idézőjelen belülre teszünk `{n}` jelölőket (`{0}`, `{1}` stb.), amik helyére a később felsorolt változóink értéke fog behelyettesítődni. A `{n}`-ben az `n` egy sorszámot jelöl, azt a sorszámot, ami szerint a szöveg után, a zárójelpáron belül felsoroljuk vesszővel a változókat. Nagyon fontos, hogy ez a sorszámozás 0-val kezdődik, vagyis a fenti példában a `{0}` az első felsorolt változó, vagyis az `a`; a `{1}` a második felsorolt változó, vagyis a `b`; a `{2}` pedig a harmadik felsorolt változó, vagyis az `ossz`.

9.3. Egy újabb elegáns módszer

Az előző módszer szépséghibája, hogy ha hosszú a szövegünk, olvasás közben nagyon nehéz átlátni, mit is fog kiírni, hiszen a tekintetünknek folyamatosan ugrálnia kell a `{n}` jelölők és a sor végén lévő változónevek között. Ezt valahogy úgy tudnánk megoldani, ha a sor vége helyett a szövegben magában helyeznénk el a változóneveket, de ezt „csak úgy simán” nyilván nem tehetjük meg, hiszen akkor sehonnan sem derülne ki, hogy az a betű egy névelő vagy egy változónév.

```
int a = int.Parse(Console.ReadLine()); // 1
int b = int.Parse(Console.ReadLine()); // 2
```

```
int ossz = a + b;                          // 3
Console.WriteLine($"{a} es {b} osszege
    {ossz}");
// 1 es 2 osszege 3
```

Programkód 13. Változók értékének kiírása – elegánsan 2.

Ha a szöveg idézőjele elé teszünk egy `$` jelet, akkor a `{}` kapcsoszárójel-párokba sorszám helyett változónevet is írhatunk. Ekkor értelemszerűen nem kell a szöveg után felsorolni újra a változókat. Ez a jelölés elsőre bonyolultnak tűnhet a sok különleges karakter miatt (dollárjel, idézőjel, kapcsos zárójel), de összességében ez eredményezi a legolvashatóbb programkódot.

10. Aritmetikai műveletek

A matematikai alapműveletek természetesen használhatók a programkódokban is, így összeadni, kivonni, szorozni és osztani minden további nélkül tudunk.

```
int a = 3, b = 4;
// egy sorban tobb azonos tipusu valtozot
    is letrehozhatunk
int osszeg = a + b; // 7
osszeg = osszeg - 1;
// osszeg erteket 1-gyel csokkentettuk,
    vagyis 6 lett
int negyzet = osszeg * osszeg; // 36
```

Programkód 14. Aritmetikai műveletek

Figyeljük meg az `osszeg = osszeg - 1;` sort, és emlékezzünk vissza, hogy az egyenlőségjel nem egyenlőséget, hanem értékadást jelent!

10.1. Az osztás és a problémák

Két egész szám összege, különbsége és szorzata is biztosan egész lesz, a hányadosuk azonban már egyáltalán nem biztosan, ez pedig okoz némi kellemetlenséget. Elsőre talán meglepő, de a legtöbb programozási nyelvben (a C# is ilyen) két egész szám hányadosa is egész lesz. Vagyis például `3 / 4` eredménye `0` lesz. Ezt egészosztásnak hívjuk, vagyis az osztási maradékkal „nem foglalkozunk”, a hányadost lefelé kerekítjük.

```
int a = 3, b = 4;
Console.WriteLine(a / b); // 0
Console.WriteLine(3 / 4); // 0
```

Programkód 15. Egészosztás

Talán még másodikra is meglepő, de két egész szám hányadosa még akkor is egész lesz, ha egy `float` típusú változóban tároljuk el a „hányadost”. (Valójában ez nem meglepő: értékadásnál először kiértékelődik az egyenlőségjel jobb oldalán található kifejezés, és utána kerül bele az eredmény a változóba. Vagyis az osztás elvégzésekor a számítógép még nem tudja, mit fogunk csinálni az eredménnyel, kiírjuk a konzolra vagy értékül adjuk egy változónak.)

```
float x = 3 / 4; // 0
```

Programkód 16. Egészosztás tört változóval

Ha azt szeretnénk, hogy ne egészosztás történjen, hanem a hagyományos értelemben vett (akár tört) hányadost kapjuk, több lehetőségünk is van. Ha konkrétan kódban leírt számot szeretnénk osztani, ha mögé tesszük, hogy `.0`, már törtként értelmezi a számítógép, és így törtként is osztja el.

```
Console.WriteLine(3.0 / 4); // 0.75
```

Programkód 17. Tört osztása

Ha azonban két `int` típusú változót szeretnénk törtként elosztani, ez az előbbi módszer nem működik. Ekkor „kézzel” kell megmondanunk a számítógépnek, hogy az osztandót `float`-ként értelmezze, úgy, hogy a változó neve elé írjuk, hogy `(float)`.

```
int a = 3, b = 4;
float x = (float)a / b; // 0.75
```

Programkód 18. Tört osztása `floart`-tá alakítással

10.1.1. Osztási maradék

Néha azonban kifejezetten az osztási maradékra lenne szükségünk. A jó hír az, hogy erre külön műveletünk van, aminek a százalékjel (`%`) a jele.

```
int maradek = 15 % 4; // 3
```

Programkód 19. Osztási maradék

10.1. Feladat. *Írjunk egy programot, amely beolvassa egy négyzet oldalát, majd kiírja a négyzet kerületét és területét.* ***Ezt a feladatot próbáld meg magadtól megoldani!***

Ha elakadnál, egy kis segítség: biztosan szükségünk lesz egy változóra, amiben az oldal hosszát tároljuk.

Megoldás. Egy lehetséges helyes megoldás:

```
Console.WriteLine("Add meg a
    negyzet oldalanak hosszat!");
float oldal =
    float.Parse(Console.ReadLine());
float kerulet = 4 * oldal;
float terulet = oldal * oldal;
Console.WriteLine("A negyzet
    kerulete {0}, terulete {1}",
    kerulet, terulet);
```

Programkód 20. Négyzet kerülete és területe

□

Egy meglepő módon rossz megoldás. Az alábbi megoldás **hibás**.

```
        Console.WriteLine("Add meg a
            negyzet oldalanak hosszat!");
        float oldal =
            float.Parse(Console.ReadLine());
        Console.WriteLine("A negyzet
            kerulete: " + oldal + oldal +
            oldal + oldal);
```

Programkód 21. Négyzet kerülete – hibás megoldás

Ha a felhasználó 3-at ad meg oldalhossznak, elsőre gondolhatnánk azt is, hogy az eredmény 12 lesz, hiszen 3+3+3+3=12, de nem.

```
        // NEM ez lesz az eredmeny:
        // A negyzet kerulete: 12
        // Hanem ez:
        // A negyzet kerulete: 3333
```

Programkód 22. Négyzet kerülete – hibás megoldás kimenete

Ennek pedig az az oka, hogy ebben az esetben a `WriteLine()` „hasában” a `+` nem számok összeadását, hanem szövegek összefűzését jelenti. Vonjuk le azt a tanulságot, hogy egy szövegbe egy változó értékét NE plusz jellel tegyük bele, használjuk inkább a `{n}`-es vagy `${}`-es formulákat. □

11. Vezérlési szerkezetek

11.1. Elágazás

Eddigi programjaink a sorokban leírt utasításokat egymás után hajtották végre. Ez azonban érezhetően kevés, ha ténylegesen azt szeretnénk, hogy a programunk reagáljon a felhasználó bemeneteire. Azt szeretnénk, hogy bizonyos utasításokat csak bizonyos esetekben hajtson végre a programunk.

Vegyük például a négyzetes feladat módosítását: olvassuk be egy téglalap két oldalát, majd döntsük el, hogy a téglalap négyzet-e. Precízebben ez azt jelenti, hogy ha a két oldal egyenlő, írjuk ki, hogy négyzet, egyébként pedig írjuk ki, hogy nem négyzet. Van tehát egy feltételünk (a két oldal egyenlő), amit kiértékelünk, és az igazságtartalmától függően különböző dolgot csinálunk. Nézzük, hogy néz ez ki programkódban.

```
int a = int.Parse(Console.ReadLine());
int b = int.Parse(Console.ReadLine());

if (a == b)
{
        Console.WriteLine("Ez egy negyzet");
}
else
{
        Console.WriteLine("Ez nem egy
            negyzet");
}
```

Programkód 23. Elágazás

Az `if` kulcsszó jelöli, hogy elágazás következik. Ezt követően egy zárójelpárba kerül a feltétel, majd egy kapcsos zárójel blokkba kerülnek az utasítások, amelyeket akkor hajt végre a program, ha a feltétel igaznak bizonyult. Ezt követheti opcionálisan az `else` kulcsszó, majd egy újabb kapcsos zárójel blokk, amibe pedig azok az utasítások kerülnek, amelyek akkor hajtódnak végre, ha a feltétel hamis.

11.1.1. Egyenlőségvizsgálat

Figyeljük meg, hogy a fenti példában az `if (a == b)` feltételben kettő egyenlőségjel szerepel. Ennek az oka az, hogy az egy egyenlőségjel az értékadást jelenti (`a = b`: a változó értéke legyen egyenlő b változó értékével), az egyenlőségvizsgálatot pedig a dupla egyenlőségjel (`a == b`: a változó és b változó értéke egyenlő-e).

11.1.2. Logikai értékek

Az `if` utáni zárójelpárban lévő kifejezés értéke egy logikai érték, vagyis igaz (`true`) vagy hamis (`false`). Ez jelenthet pl. valamilyen relációt (`<`, `>`, `<=` (kisebb vagy egyenlő), `>=` (nagyobb vagy egyenlő), `==` (egyenlő), `!=` (nem egyenlő)).

A logikai értékekkel természetesen végezhetünk logikai műveleteket: és (jele: `&&`), vagy (jele: `||`), kizáró vagy (jele: `^^`), negálás (jele: kifejezés elé írt `!`). Összetett logikai kifejezések esetén mindenképp érdemes zárójeleket használni, mert a logikai műveletek precedenciája (negálás, és, vagy) nem mindig triviális első olvasásra.

Logikai kifejezés	A kifejezés értéke
`1 < 2`	`true`
`2 < 2`	`false`
`a < 2`	`a` értékétől függ
`1 == 1`	`true`
`1 != 2`	`true`
`1 != 1`	`false`
`true && false`	`false`
`true || false`	`true`
`!true`	`false`

11.2. Számláló ciklus

Ha visszakanyarodunk oda, hogy alapesetben a programunk utasításai egymás után hajtódnak végre, beláthatjuk, hogy az elágazás lényegében azt a lehetőséget adta meg nekünk, hogy végrehajtás nélkül átugorjunk egy programrészt (ha a feltétel igaz, az `else` ágat, ha a feltétel hamis, az `if` ágat ugorjuk át végrehajtás nélkül). Ez egy nagyon gyakran használt, és hasznos vezérlési szerkezet, de korántsem elegendő.

A való életben is gyakran hajtunk létre nagyon hasonló dolgokat sokszor egymás után. Az eddig megismert programozási eszköztárral pl. egy olyan egyszerű feladatot sem igazán tudnánk végrehajtani, hogy kérjünk be a felhasználótól egy számot, majd írjuk ki annyiszor, hogy „szia”. Vagy ugyanígy bajban lennénk, ha el kellene számolnunk 1-től x-ig. Ezekben a feladatokban az a közös, hogy

ugyanazt a dolgot szeretnénk sokszor végrehajtani egymás után. (A kötekedő kedvű olvasó most felkaphatná a fejét, hogy nem igaz, ez nem közös bennük, hiszen a második példában nem ugyanazt a valamit hajtjuk végre egymás után, pedig valójában de. Ez a valami pedig az az utasítássorozat, hogy megnöveljük a számot és kiírjuk. Fontos tehát átállítani az agyunkat a tekintetben, hogy „ugyanaz" alatt programozásban már olyan dolgokat is érthetünk, amik valaminek a függvényében csinálják „ugyanazt.")

A megoldás tehát egy olyan vezérlési szerkezet lenne, amely nem átugrani tud egy végre nem hajtandó programrészt, hanem épp ellenkezőleg: újra végre tud hajtani egy programrészt, vagyis képes „visszafele" ugrani a programunkban. Ez a vezérlési szerkezet pedig nem más, mint a ciklus.

Többféle ciklus is létezik, mi most elsőként a számláló (`for`) ciklussal ismerkedünk meg. Ennek a szintaktikája a következő:

```
for (int i = 0; i < length; i++)
{
        ...
}
```

Programkód 24. Számláló ciklus

Tipp: a programozók közismerten lusták, néhány tipikus dolgot viszont nagyon gyakran kell(ene) leírni. A Visual Studio bizonyos rövidítések beírását követő két <TAB> után leírja helyettünk ezeket a sorokat. Például a fenti `for` ciklus szerkezetet a `for` <TAB> <TAB> beírásával kaphatjuk meg, de elágazást (`if` <TAB> <TAB>) és konzolra kiírást (`cw` <TAB> <TAB>) is generálhatunk így.

Ismerkedjünk meg a `for` ciklussal! A legelején a `for` kulcsszó áll, ezt illendő szóköznek követnie, majd egy zárójelpárban két pontosvessző osztja három részre a ciklus működésének leírását. Ezt egy `{ }` blokk követi. A for utáni zárójelek között fogjuk leírni, hogyan működik a ciklusunk, a `{ }` blokkban pedig azt, hogy mit csinál.

Nézzük, mi szerepel a `for` utáni zárójelek között (pontosvesszőkkel tagolva):

- `int i = 0;`

  Ez a kódrészlet már ismerős kell, hogy legyen: létrehozunk egy egész (`int`) típusú, `i` nevű változót, és a `0` kezdőértéket adjuk neki. Ezt a változót **ciklusváltozó**nak nevezzük. Fontos, hogy ez a változó csak a cikluson belül érhető el, a ciklusból való kilépés után már nem. A ciklusváltozóknak a kontextusukban egyedieknek kell lenniük, de két egymást követő ciklus ciklusváltozója már lehet azonos nevű (hiszen mindkét változó csak a saját ciklusán belül „létezik".)

  Amennyiben egyszerre több ciklusváltozóra van szükségünk (lásd később), azokat i-től kezdődő kisbetűkkel szokás elnevezni (i, j, k, ...) Ez persze azt is jelenti, hogy nem-ciklusváltozókat nem illik így elnevezni az átláthatóság érdekében.

- `i < length;`

  Ez a kódrészlet sem ismeretlen: ez egy ugyanolyan logikai kifejezés, mint ami az elágazás (`if`) zárójelpárjában is szerepelt. Ez a ciklusunk **ciklusfeltétel**e (más néven **leállási feltétel**e), ami azt jelenti, hogy a ciklusunk

addig fog futni (addig fogja ismételni magát), amíg ez a feltétel igaz. Ha hamissá válik, a programunk kilép a ciklusból.

Természetesen az `i < length` önmagában helytelen, nem is indul el így a programunk. A length helyére leggyakrabban egy egész számot vagy egy egész típusú változónevet írunk. Egyelőre csak megjegyezzük, de fontos, hogy a ciklus ciklusfeltétele nemcsak `i < length` alakú logikai kifejezés lehet, hanem bármilyen logikai kifejezés.

- `i++`

  Ez pedig az a kódrészlet, ami a ciklus minden végrehajtódása után lefut. Jelen esetben `i` értékét megnöveli `1`-gyel. Az `i++` az `i = i + 1` és az `i +=1` tömörebb alakja, mindhárom utasítás ugyanazt csinálja.

- `{ ... }`

  A `{ }` blokkba írt utasításokat a **ciklus törzsé**nek vagy **ciklusmag**nak nevezzük. *(A ciklus(mag) egy lefutását egy **iterációnak** nevezzük.)*

Most, hogy ízekre szedtük a ciklust, rakjuk össze az építőkockákat.

1. Először létrejön a ciklusváltozónk a kezdeti értékével.
2. A program kiértékeli a leállási feltételt.
   (a) Ha a feltétel igaz:
      i. Végrehajtódik a `{ }` blokkban lévő kód.
      ii. Lefut a ciklusváltozó értékét módosító kód (`i++`)
      iii. Visszalépünk a 2. pontra (feltételvizsgálat)
   (b) Ha a feltétel hamis, a programunk kilép a ciklusból.

A fentiekből következik néhány fontos tulajdonság:

- A leállási feltétel a ciklus minden lefutása után újra és újra kiértékelődik.
- A leállási feltétel első kiértékelése a ciklusmag első lefutása **előtt** történik meg.
- Vagyis minden további nélkül lehetséges, hogy a ciklusmag egyszer sem fog lefutni.
- Ahogy arra is figyelnünk kell, hogy a ciklusunk leálljon valamikor. Egyébként végtelen ciklusba kerül a programunk, vagyis használhatatlan lesz, „lefagy”.

A korábban említett egyszerű feladatot, hogy olvassunk be a felhasználótól egy számot, és írjuk ki annyiszor, hogy „szia”, most már könnyedén meg tudjuk oldani.

```
int ennyiszer =
    int.Parse(Console.ReadLine());
for (int i = 0; i < ennyiszer; i++)
{
        Console.WriteLine("szia");
}
```

Programkód 25. Számláló ciklus példa

Fontos megérteni, hogy ha a felhasználó 10-et ír be, a fenti ciklus miért 10-szer írja ki, hogy „szia”. Az `i` változó értéke 0-ról indul, és minden kiírás után növekszik.

`i` értéke	`i < 10`	
0	`true`	kiírás
1	`true`	kiírás
2	`true`	kiírás
3	`true`	kiírás
4	`true`	kiírás
5	`true`	kiírás
6	`true`	kiírás
7	`true`	kiírás
8	`true`	kiírás
9	`true`	kiírás
10	`false`	kilépés a ciklusból

Vagyis a ciklus törzsében i értéke [0,9] értékeket vesz fel, vagyis a 10-et már nem.