Utasítások, változók és típusok, elágazások

Összefoglaló

Utasítások

Egy számítógépes program utasítások sorozatából áll, amelyek lépésről lépésre meghatározzák a program működését. A C# nyelvben az egyszerű utasítások mindig pontosvesszővel végződnek. Az összetett utasítások több egyszerű utasítást vagy bonyolultabb műveletet tartalmaznak, és ezeket kapcsos zárójelek határolják, amelyek a kódblokk kezdetét és végét jelölik ki.

A parancssorra (terminal, console) történő kiírásra használhatjuk a **Console.Write** és **Console.WriteLine** utasításokat. Az utasítás nevét követően zárójelek között kell megadnunk, mit szeretnénk kiírni a képernyőre. Ha például egy rögzített szöveget akarunk kiírni, akkor ezt idézőjelek közé kell tennünk.

```
Console.WriteLine("Hello, World!");
```

Az alapértelmezett bemenetről (ez általában a billentyűzetet jelenti) történő adatbeolvasáshoz használhatjuk a **Console.ReadLine** utasítást.

```
Console.ReadLine();
```

Ha egy utasítás figyelmen kívül hagyását kérjük a végrehajtás során, vagy utasításként nem értelmezhető szöveget szeretnénk a forráskódba írni, úgynevezett megjegyzéseket (kommenteket) használhatunk.

```
// Console.WriteLine("Ez nem lesz kiírva a parancssorban.");
// Ezt sem vesszük figyelembe végrehajtáskor
```

Változók és típusok

Egy programban szinte mindig szükség van adatok tárolására, amit változók segítségével oldhatunk meg. A változók a program futása alatt átmenetileg vagy tartósan tárolják az adatokat. Minden változónak van egy típusa, amely meghatározza, hogy milyen jellegű adatot tárolhatunk benne. A változó használatának első lépése a deklarálás, ahol megadjuk a változó típusát és nevét (azonosítóját). Miután a változót deklaráltuk, a programban ezen a néven hivatkozhatunk rá, és értéket rendelhetünk hozzá, amit értékadásnak nevezünk. Az értékadást az $=$ operátorral végezhetjük el: az operátor bal oldalán álló változó kapja meg az operátor jobb oldalán álló kifejezés értékét. Ezt követően a változó értékét lekérdezhetjük és kiértékelhetjük.

```
int x;     // deklaráció
x = 1;     // értékadás
int y = 2;     // a deklaráció és értékadás összevonható
Console.WriteLine(x);     // kiértékelés
```

A változó típusának megválasztásakor arra kell figyelnünk, hogy az illeszkedjen a tárolandó adathoz. Ez sok esetben egyértelmű, de például egy egész számot eltárolhatunk akár szöveges formában is egy karakterlánc

típusú változóban, miközben valószínűleg nem ez a leghatékonyabb megoldás. A forráskódban megjelenő konkrét értékek, amelyeket közvetlenül a kódba írunk, az úgynevezett literálok. A literálok típusát gyakran az alakjukból lehet felismerni.

```
int egesz = 123;    // 32-bites elöjeles egész
double valos = 1.23456;    // 64-bites lebegöpontos szám
bool logikai = true;    // logikai érték
char karakter = '\n';    // egyetlen karakter (itt épp a sortörés karakter)
string karakterlanc = "Vincent! We happy?";    // szöveg
```

Típuskonverzió, típuskényszerítés

Gyakoran előfordul, hogy adatokat különböző típusok között kell átkonvertálni, hogy a program helyesen működjön. A C# nyelv *erősen típusos*, ami azt jelenti, hogy egy változó értékadásakor csak annak típusával egyező (vagy azzal kompatibilis) értéket rendelhetünk hozzá. A típuskonverzió lehetővé teszi, hogy egy adott típusú adatot valamilyen módon egy másik típusba alakítsunk át (például egy számot szöveggé, vagy fordítva). A típuskonverzió C#-ban két módon történhet.

Implicit típuskonverzióról beszélünk, amikor az átalakítás automatikusan történik, vagyis nem kell külön jeleznünk, hogy típusátalakítást kérünk. Ilyenkor az egyik típus egy másik típusba konvertálható anélkül, hogy az adatvesztés lehetősége fennállna. Erre elgéséges feltétel, ha a konvertálandó típus értéktartománya szűkebb a céltípus értéktartományánál.

```
int egesz = 13;
double valos = egesz;    // a változó automatikusan lebegöpontossá konvertálható
```

Explicit típuskonverzióról beszélünk, amikor a konverzió nem történik meg automatikusan, és a programozónak explicit módon kell jeleznie, hogy átalakítást kér. Ez olyan esetekben szükséges, amikor az átalakítás során fennáll az adatvesztés lehetősége, vagyis a program létrehozója ennek tudatában, a kódja ismeretében kéri az átalakítást. Explicit típuskonverziót, más néven *típuskényszerítést* (typecast) válthatunk ki azzal, hogy a konvertálandó változó vagy érték bal oldalára, kerek zárójelek közé írjuk a céltípus megnevezését.

```
double egyikValos = 13.0;
double masikValos = 13.5;
int egyikEgesz = (int)egyikValos;    // a konvertált érték 13, nincs adatvesztés
int masikEgesz = (int)masikValos;    // a konvertált érték 13, adatvesztés történt
```

Szintén gyakori eset, amikor valamilyen adatot szöveges formátumra alakítunk, vagy szöveges adatból szeretnénk kinyerni például numerikus értéket. A **ToString** utasítással bármely értéket szöveges típussá alakíthatunk.

```
double szam = 123.456;
string szoveg = szam.ToString();    // az eredmény "123.456" szöveges formában
char egyBetu = 'A';
string masikSzoveg = egyBetu.ToString();    // az eredmény "A" szövegként
```

Megfelelő formátumú szövegből a benne lévő érték kinyerhető a céltípushoz tartozó **Parse** utasítással. Helytelen formátum esetén (például tizedespont helyett tizedesvesszőt tartalmazó szöveges formátumú valós érték) a program futásidejű hibával áll le.

```
string egeszSzovegkent = "13";
int x = int.Parse(egeszSzovegkent);    // x értéke 13 lesz
string logikaiSzovegkent = "false";
bool y = bool.Parse(logikaiSzovegkent);    // y értéke false lesz
string hibasFormatum = "13 és egy kicsi";
double z = double.Parse(hibasFormatum);    // futásidejü hibát okoz
```

A program vezérlése elágazásokkal

A programkódba írt elágazások használatával döntéseket hozhatunk, és különböző ágakon lévő utasítások végrehajtását indíthatjuk el attól függően, hogy egy adott feltétel igaz vagy hamis. Elágazást az `if` kulcsszóval képezhetünk, amelyet egy zárójelek közé írt logikai kifejezésnek (feltételnek) kell követnie. Amennyiben a feltétel teljesül (igazra értékelődik ki), a feltételt követő utasítás (egyszerű vagy összetett) lefut, ellenkező esetben a végrehajtás átugorja az utasítást. Jó gyakorlat, hogy akkor is összetett utasítást (kódblokkot) adjunk meg, ha az csak egyetlen utasítást tartalmaz, így elkerülhetjük, hogy a szintaktika miatt több utasítás közül csak az első hajtódjon végre, ezzel nem várt működést kiváltva.

Az alábbi kódrészlet például csak abban az esetben írja felül az `ertek` változó tartalmát, ha az nagyobb, mint `maxErtek`.

```
int ertek = 13;
int maxErtek = 100;
if (ertek > maxErtek)
{
    ertek = maxErtek;
}
```

Lehetőségünk van mind a feltétel igaz ágát, mind a hamis ágát kezelni az `else` kulcsszóval. Az alábbi kódrészlet a felhasználó által megadott névtől függően írja ki a a két lehetséges üzenet egyikét.

```
string jatekosNev = Console.ReadLine();
if (jatekosNev == "The Doom Slayer")
{
    Console.WriteLine("Rip and Tear, until it is done.");
}
else
{
    Console.WriteLine("You were nothing but a usurper -- a false idol.");
}
```

Több feltétel ellenőrzésére is lehetőségünk van az `if` és `else` kulcsszavak megfelelő kombinálásával. Ilyen esetben az első igaz feltételt követő utasítás, ennek hiányában a „különben-ág" fut le.

```
int szam = -13;
if (szam < 0)
{
    Console.WriteLine("Negatív érték.");
}
else if (szam == 0)
{
    Console.WriteLine("Nulla.");
}
else
{
    Console.WriteLine("Pozitív érték.");
}
```

Operátorok

Az operátorok olyan speciális szimbólumok vagy kulcsszavak, amelyek egy vagy több operanduson végeznek műveleteket, és egy eredményt adnak vissza. Az operátorok segítségével különböző műveleteket hajthatunk végre, például összeadást, kivonást, logikai műveleteket, értékadást, összehasonlítást. A teljesség igénye nélkül az alábbi operátorokat használjuk leggyakrabban.

Az *aritmetikai operátorokat* matematikai műveletek elvégzésére használjuk. A `+`, `-`, `*`, `/` szimbólumokkal rendre két szám összegét, különbségét, szorzatát és hányadosát képezhetjük. A `%` operátorral maradékképzést (maradékos osztást) valósíthatunk meg. Az operátorok a két operandus típusával egyező eredményt adanak vissza (vagyis például két egész szám hányadosa minden esetben egész számként jelenik meg).

Az *összehasonlító operátorok* két érték összehasonlítására szolgálnak, logikai eredményt adnak vissza. Két érték egyenlőségét a `==` szimbólummal (dupla egyenlőségjellel) vizsgálhatjuk, a nem-egyenlőséget a `!=` szimbólummal. A `<`, `<=`, `>=`, `>` relációs operátorok a megszokott összehasonlításokat valósítják meg.

A *logikai operátorok* két logikai érték kombinálására szolgálnak. A `NEM` logikai operátort a `!` szimbólummal érjük el. A `(!a)` kifejezés értéke pontosan akkor igaz, ha `a` hamis. Az `ÉS` logikai operátort a `&&` szimbólummal érjük el. Az `(a && b)` kifejezés értéke pontosan akkor igaz, ha `a` és `b` egyszerre igaz, minden más esetben a kifejezés értéke hamis. A `VAGY` logikai operátort a `||` szimbólummal érjük el. Az `(a || b)` kifejezés értéke pontosan akkor hamis, ha `a` és `b` egyszerre hamis, minden más esetben a kifejezés értéke igaz.

Változók értékének beállítására használjuk az *értékadó operátort*. A már említett módon értékadást az `a = b;` utasítással valósíthatunk meg, melynek hatására az `a` változó értéke a `b` kifejezés értékét veszi fel.

Feladatok

1 Hozzunk létre egy új parancssoros projektet, majd írjunk programot, amely megjeleníti a `Hello, World!` szöveget a képernyőn. Az alkalmazás az Enter billentyű leütésekor érjen véget.

2 Bővítsük ki az előbbi programot: helyezzünk el néhányat az alábbi utasításokból a szöveg megjelenítését végző utasítás elé vagy után. Az utasítások a kurzor és a parancssori ablak jellemzőit módosítják.

- `Console.Clear()`
- `Console.WindowHeight`
- `Console.WindowWidth`
- `Console.BackgroundColor`
- `Console.ForegroundColor`
- `Console.SetCursorPosition()`
- `Console.CursorVisible`

3 Készítsünk programot, amely elkéri a felhasználó nevét, majd név szerint köszönti őt.

4 Készítsünk programot, amely elkéri a felhasználó születési évét, ez alapján pedig kiszámítja és kiírja az életkorát. A program írja ki azt is, hány éves lesz a felhasználó a következő évben.

5 Készítsünk programot, amely elkéri a felhasználó testmagasságát (h, méterben) és testtömegét (m, kilogrammban), majd kiszámítja és kiírja a felhasználó testtömegindexét (BMI). A számításhoz használt formula:

$$BMI = \frac{m}{h^2}$$

6 Kérjünk el a felhasználótól egy másodpercben megadott időtartamot, majd írjuk ki azt `perc:másodperc` formátumban.

Példa

```
Az időtartam másodpercben: 123
Az időtartam formázva: 2:03
```

7 Kérjük el a felhasználótól a jelszavát, majd kérjük el még egyszer megerősítésképp. Ha egyezik a két megadott jelszó, nyugtázzuk egy zöld színnel kiírt üzenettel, ellenkező esetben jelenítsünk meg egy piros színű hibaüzenetet.

8 Alakítsuk át úgy az előző programot, hogy a jelszó beírásakor a karakterek helyett csak *-ok jelenjenek meg. Legyen lehetőség tévesen bevitt karakter törlésére is a Backspace billentyűvel.

9 Kérjünk el a felhasználótól kettő számot és egy műveleti jelet, majd írjuk ki a képernyőre az adott művelet eredményét a két szám között elvégezve.

Példa

```
Add meg az első számot: 3
Add meg a második számot: 7
Add meg a műveletet: *
3 * 7 = 21
```

10 Alakítsuk át úgy az előző programot, hogy az képes legyen helyesen formázott matematikai kifejezések kiértékelésére. Első lépésként valósítsuk meg, hogy a beolvasott szöveg alapján bármely kétoperandusú műveletet (összeadás, kivonás, szorzás, osztás) el tudja végezni a program.

Példa

```
A kiszámítandó kifejezés: 18 / 3
Eredmény: 6
```

A fordított lengyel jelölés (Reverse Polish Notation) használatával valósítsuk meg, hogy bármely helyesen zárójelezett, a fenti műveleteket tartalmazó kifejezést ki tudjon értékelni a program.

Példa

```
A kiszámítandó kifejezés: 1 + (18 / 4) * (2 - (4 * 7))
Eredmény: -103
```

11 Kérjünk egy a felhasználótól egy 0 és 9 közötti értéket, majd írjuk ki a számot szövegesen. Ha a tartományon kívüli értéket ad meg, tájékoztassuk hibaüzenettel.

Példa

```
Adj meg egy számot: 9
Az általad megadott szám: kilenc
```

12 Kérjünk egy a felhasználótól egy betűt, majd írjuk ki, hogy magánhangzót vagy mássalhangzót adott-e meg.

13 Adott egy V térfogatú tartály, amit két csővezetéken keresztül töltünk fel fel. Ismerjük a vezetékekben a térfogatáramot (az egy óra alatt átfolyó térfogatot). A két vezetéket egyszerre nyitjuk meg, majd T óráig folyni hagyjuk. Adjuk meg, hogy az időtartam végén mennyire telt meg a tartály.

Példa

```
V = 1000
R1 = 100
R2 = 120
T = 3
--> A tartály 66%-ban lesz tele.
```

Példa

```
V = 100
R1 = 100
R2 = 100
T = 2.5
--> A tartály 400 m3-rel lesz túltöltve.
```