Ciklusok használata, véletlen számok, matematikai függvények

Összefoglaló

Feltételes ciklusok

A ciklusok lehetővé teszik, hogy egy adott műveletsorozatot többször végrehajtsunk, anélkül, hogy ugyanazokat az utasításokat újra és újra meg kellene írnunk. A ciklus használatakor meg kell adnunk valamilyen feltételt, amelytől függően a ciklushoz rendelt utasítások vagy megismétlődnek, vagy a ciklus véget ér, és a végrehajtás a ciklus után folytatódik.

A *feltételes* ciklus lényege, hogy egy megadott logikai feltétel igaz értéke esetén újra és újra végrehajtjuk a ciklushoz tartozó utasításokat (ezt szokás *ciklusmagnak* is nevezni), amíg a feltétel hamis nem lesz. Ha a feltétel a ciklus elején hamis, akkor a ciklusmag egyszer sem fog lefutni.

Az alábbi kódrészletben nulla kezdőértékkel inicializálunk egy változót, majd azt a ciklusmagban kiírjuk és megnöveljük mindaddig, amíg a változó értéke el nem éri a 10-et.

```
int i = 0;
while(i < 10)
{
    Console.WriteLine(i);
    i = i + 1;
}
```

Végtelen ciklusról beszélünk, ha a ciklusfeltétel soha nem válik hamissá. Ha például egy ciklusfeltétel olyan változótól függ, amelynek értéke a ciklusmag hatására nem változik meg, és a ciklusfeltétel kezdetben igaz, akkor végtelen ciklust kapunk. A végtelen ciklus használata néhány esetben tudatos döntés, például mikrovezérlők[1] esetén általánosan alkalmazott gyakorlat.

A feltétel kiértékelése történhet a ciklusmag előtt (ilyenkor szokás *elöl tesztelő* ciklusról beszélni), illetve a ciklusmagot követően is (ilyenkor *hátul tesztelő* ciklusként szokás rá hivatkozni). Utóbbi esetben a ciklusmag utasításai legalább egyszer végrehajtásra kerülnek, amely bizonyos esetekben hasznos lehet.

Az alábbi példában deklarálunk egy változót, amely azonban nem kap kezdőértéket a ciklus előtt. Mivel `do-while` ciklust használunk, így a ciklusmag legalább egyszer biztosan lefut, ahol viszont a változó értéket kap, tehát a ciklusfeltétel a ciklus végén biztosan kiértékelhető.

```
string input;
do
{
    input = Console.ReadLine();
}
while (input != "stop");
```

[1] Konkrét feladat végrehajtására tervezett, tipikusan apró méretű és egyszerű felépítésű számítógép.

Számláló ciklusok

A *számláló ciklust* olyan esetekben célszerű használni, amikor a ciklusmagot ismert számú alkalommal szeretnénk végrehajtani. Ehhet általában egy számlálót használunk, amelynek értéke minden iteráció után változik, amíg el nem éri a meghatározott határt.

C#-ban a `for` ciklus három tagból áll:

- Inicializátor: ebben deklaráljuk a ciklus változóját, amit kezdőértékkel látunk el. Ez a rész csak egyszer fut le, mielőtt a ciklus elindul.
- Feltétel: ez határozza meg, hogy a ciklusmag hány alkalommal fusson le. Amíg a feltétel igaz, a ciklus folytatódik. A ciklusmag minden lefutása előtt kiértékelődik.
- Iterátor: a ciklusmag minden lefutása után végrehajtódik. Általában a számláló változó növelése vagy csökkentése történik (egyébként tetszőleges műveletet végrehajthatunk).

Az alábbi kódrészlet a `for` ciklus használatára mutat példát, eredménye pontosan megegyezik a fentebb `while` ciklussal már megvalósított nullától tízig számlálással. Az inicializátor részben most is egy változót hozunk létre, ami a nulla kezdőértéket kapja. A feltétel akkor igaz, ha a változó értéke tíznél kisebb. Az iterátor részben a változó értékét növeljük meg eggyel.

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

Véletlenszám-generálás

Álvéletlen számok alatt olyan, általában számítógéppel előállított számokat értünk, amelyek véletlenszerű mintázatot mutatnak, de valójában egy algoritmus (vagyis egy pontosan ismert, *determinisztikus* szabály) hozza létre őket. Használatuk széleskörűen elterjedt különféle tudományos, mérnöki és gazdasági szimulációs eljárásokban, de ugyanígy kriptográfiai problémákban és videójátékokban is találkozunk velük.

C#-ban véletlen számok generálásához használhatjuk a beépített `Random` típust, amellyel előállíthatunk egész és lebegőpontos számokat is. A véletlen érték előállítása előtt létre kell hoznunk az azokat generáló objektumot az alábbi utasítással.

```
Random generator = new Random();    // a változó neve tetszöleges lehet
```

A véletlen-generátor objektumot ezt követően használhatjuk arra, hogy egyenletes eloszlású[2] véletlen számokat állítsunk elő a segítségével.

Az alábbi példában az első utasítás hatására egy véletlen számot kapunk nulla és a lehetséges legnagyobb 32-bites egész szám között úgy, hogy az előálló érték biztosan kisebb a felső határnál (2^{31}-nél). A második utasításban egy felső határt adhatunk meg, ekkor egy nullánál nem kisebb, de a felső határnál biztosan kisebb értéket kapunk. A harmadik utasítással egy megadott alsó és felső határ közötti véletlen értéket kapunk, ahol az alsó határ előfordulhat a kapott számok között, de a kapott értékek minden esetben kisebbek a felső határnál.

[2] Egy véletlen változót egyenletes eloszlásúnak hívunk, ha bármely lehetséges értéke ugyanakkora eséllyel fordul elő.

```
int x = generator.Next();           // egész szám
int y = generator.Next(13);         // egész szám 0 és 12 között
int z = generator.Next(69, 420);    // egész szám 69 és 419 között
```

Egy 0 és 1 közötti egyenletes eloszlású lebegőpontos értéket kapunk az alábbi utasítással (a kapott értékek most is biztosan kisebbek 1-nél).

```
double randomNumber = generator.NextDouble();
```

Matematikai függvények

C#-ban a matematikai függvényeket a `System.Math` osztályon keresztül érjük el. Az teljesség igénye nélkül megemlítjük a fontosabb aritmetikai műveleteket, mint például a `Math.Abs()` utasítást az abszolútérték kiszámításához, a `Math.Pow()` utasítást a hatványozáshoz, valamint a `Math.Sqrt()` utasítást a négyzetgyök-vonáshoz. A `Math.Log2()`, `Math.Log()` és `Math.Log10()` utasítások rendre a 2-es, természetes és 10-es alapú logaritmusok kiszámítását teszik lehetővé. Az utasítások (függvények) paramétere és az általuk visszaadott érték ugyanolyan típusú, általában egy lebegőpontos érték.

```
double x1 = Math.Abs(-1.23);  // a paraméter és az eredmény típusa lebegöpontos érték
int x2 = Math.Abs(-1);        // a paraméter és az eredmény típusa egész érték
double y1 = Math.Pow(1.2, 3.4);    // kiszámítja 1.2-nek a 3.4-ik hatványát
double y2 = Math.Sqrt(123.4);      // kiszámítja 123.4 négyzetgyökét
double y3 = Math.Log10(42);        // kiszámítja 42 10-es alapú logaritmusát
```

Hasonlóan használhatóak a trigonometriai függvények is. Fontos megemlíteni, hogy ezeknek a paraméterét azonban nem fokban kifejezett szögként, hanem ívmértékként (radiánban kifejezett valós számként) kell megadnunk. A fontosabb matematikai konstansok, mint a π és az e értéke a `Math.PI` és a `Math.E` utasításokkal érhető el. Megjegyezzük, hogy ezek nem függvények, így nem kell zárójeleket kitennünk a nevük után.

```
double z1 = Math.Cos(0);              // kiszámítja 0 rad koszinuszát
double z2 = Math.Sin(Math.PI / 2);    // kiszámítja 90° szinuszát
double z3 = Math.Tan(1.23);           // kiszámítja 1.23 rad tangensét
```

Feladatok

1 Készítsünk programot, amely elkér a felhasználótól egy N pozitív egész számot, majd kiírja az egész számokat 0 és N között. Módosítsuk úgy a programot, hogy az csak a páros számokat írja ki.

2 Tároljuk egy változóban a felhasználó jelszavát. Addig kérjük el tőle a jelszót a parancssorról, amíg az nem egyezik az eltárolttal. Módosítsuk úgy a programot, hogy a felhasználó három sikertelen próbálkozás után kapjon hibaüzenetet.

3 Írjunk programot, amely addig generál véletlen számokat 1 és 1000 között, amíg az meg nem egyezik a program kezdetén a felhasználó által megadott számmal. Számoljuk meg, hány próbálkozás kellett a találathoz.

4 Társasjátékoknál gyakori, hogy az kezd, aki először hatos dob. Készítsünk egy alkalmazást, amely eldönti, hogy N játékos közül ki kezdjen. Minden játékosnál az `Enter` leütésére dobjunk egy véletlen számot 1 és 6 között, majd ha az nem hatos, ugorjunk a következő játékosra. Ha körbeértünk, a folyamat induljon újra, egészen addig, amíg valaki hatost nem dob.

5 Írjunk programot, amelynek kezdetén adott egy pozitív egész szám, a „gondolt szám". A felhasználónak ki kell találnia, hogy mi a gondolt szám. Ehhez a felhasználó megadhat számokat, melyekről a program megmondja, hogy a gondolt számnál nagyobbak vagy kisebbek-e. A program akkor ér véget, ha a felhasználó kitalálta a gondolt számot. A program jelenítse meg a felhasználó próbálkozásainak számát is.

6 Kérjünk el a felhasználótól egy N pozitív egész számot, majd írjuk ki az alábbiakat:

- N páros vagy páratlan
- N valódi pozitív osztóinak száma (1-et és N-et nem kell beleszámolnunk)
- N prímszám vagy összetett szám

7 Kérjünk el egy pozitív egész számot, majd írjuk ki a faktoriálisát.

$$N! = N \times (N-1) \times (N-2) \times \ldots \times 3 \times 2 \times 1$$

Példa

```
N = 5
5! = 5×4×3×2×1 = 120
```

8 Készítsük programot, amely kiírja a képernyőre a szorzótáblát az alábbihoz hasonlóan.

	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

9 Készítsünk időzítő alkalmazást, amely elkér egy másodpercben megadott időtartamot, majd kiírja azt a képernyőre `perc:másodperc` formátumban, és visszaszámlálást indít. Minden eltelt másodperc után törölje a képernyőt, és írja ki a még hátralévő időt. A visszaszámlálást végét jelezze a képernyő pirosra váltásával és sípolással.

A késleltetéshez használjuk a `System.Threading.Thread.Sleep(1000);` utasítást.

10 Készítsünk tízes számrendszerből kettes számrendszerbe átváltó alkalmazást. A bemenet legyen egy 32 bites előjel nélküli egész (`uint`), kimenetként pedig jelenítsük meg az érték kettes számrendszerbeli alakját 8 bites blokkokban, big endian formátumban.

Példa

```
420 (10) = 00000000 00000000 00000001 10100100 (2)
```

11 Készítsünk egyszerű félkarú rabló játékot. A játék elején a játékos 100 kredittel rendelkezik, a tét alapesetben 1 kredit. A `Spacebar` billentyű lenyomásakor a játék három véletlen számjegyet pörget. Két egyforma szám esetén a tét 10-szeresét, három egyforma esetén a tét 50-szeresét nyeri a felhasználó. Pörgetés előtt a tétet a `Fel` és `Le` kurzorbillentyűkkel lehet módosítani. A játék véget ér `Escape` nyomáskor, vagy ha a játékosnak elfogy a kreditje.

12 Egészítsük ki az előző félkarú rabló játékot *ASCII art* grafikai elemekkel: a számok helyett karakterekből kialakított színes figurák (pl. pikk, kőr, treff, káró) jelenjenek meg pörgetéskor.

13 Egy új kriptovaluta árfolyamának alakulását szimuláljuk. Jelölje az aktuális árfolyamot P_t (valós szám). A kriptovaluta árfolyamát a következő órában a

$$P_{t+1} = r \times P_t + \varepsilon_t$$

képlettel modellezzük, ahol r egy adott paraméter, ε_t pedig egy véletlen valós szám a $[-\alpha, \alpha]$ intervallumból. Írjuk ki a képernyőre az árfolyam alakulását különböző r és α értékekkel a felhasználó által megadott számú órára.