Tömbök és listák, bejáró ciklus

Összefoglaló

Tömbök

A tömbök segítségével azonos típusú adatokból tudunk egyszerre többet egyetlen változón keresztül kezelni. A tömb egy olyan gyűjtemény, ahol az egyes elemekre azok sorszámával vagy indexével hivatkozhatunk. Az indexelés a C# programnyelvben – és a programnyelvek többségében – nullától indul.

Hasonlóan a változóknál megismertekhez, első lépésként deklarálnunk kell a tömb elérésére használt változót, és létre kell hoznunk a tömböt, amihez rögzítenünk szükséges a tömb elemeinek típusát, valamint a tömb méretét (a maximálisan elhelyezhető elemek számát). A deklaráció és a tömb létrehozása (inicializáció) elvégezhető egy lépésben. Lehetőségünk van tömb létrehozására az elemeinek felsorolásával is.

```
bool[] myBools;                 // tömb változó deklarációs
myBools = new bool[2];          // tömb létrehozása 2 elemmel
int[] myNums = new int[10];     // a deklaráció és létrehozás összevonható
char[] someChars = { 'a', 'b', 'c' };    // tömb létrehozás az elemek felsorolásával
```

Ha nem felsoroljuk a tömb elemeit, hanem egy megadott méretű „üres tömb" létrehozását kérjük, akkor a tömb a típusának megfelelő alapértelmezett értékekkel lesz feltöltve. Ez egész számok esetében 0, lebegőpontos számok esetében 0.0, karakternél `'\0'` (null karakter), logikai típusnál `false`, referencia típusoknál (pl. string, objektumok) `null` (üres referencia).

Ezt követően értéket adhatunk vagy lekérhetjük egy adott indexen lévő elemét a tömbnek az *indexelő operátor* (szögletes zárójelpár) segítségével.

```
myBools[1] = true;              // tömbelem értékadása
bool someBool = myBools[0];     // tömbelem értékének lekérése
```

A tömb mérete lekérdezhető a `Length` utasítással. A nulla-alapú indexelésből következően a legnagyobb lehetséges index értéke `Length` $- 1$, ezt túllépve indexelési hibával áll le a program futása.

```
int count = myNums.Length;    // count értéke 10 lesz
```

Többdimenziós tömbök

Kétdimenziós tömbök segítségével táblázatos (mátrix-szerű) adatokat tárolhatunk, ahol a tömb egy elemének eléréséhez kettő indexet (nevezhetjük sor- és oszlopindexnek) kell megadnunk. Ezt általánosítva kapjuk a többdimenziós tömböket, ahol egy N-dimenziós tömb egy elemének elérése N darab index segítségével lehetséges.

```
string[,,] mysteriousArray = new string[51, 33, 23];
mysteriousArray[9, 11, 13] = "The Truth is out there, Mulder. But so are lies.";
```

Többdimenziós tömbök esetén a `Length` utasítással a teljes tömb elemszámát kapjuk meg. Ha egy adott dimenziója mentén vett elemszámra van szükségünk, azt a **GetLength** utasítással kaphatjuk meg, amelynek meg kell adnunk, hányadik dimenzióhoz tartozó méretet adja vissza.

Az alábbi kódrészletben először létrehozunk egy üres 11×13-as, egészekből álló kétdimenziós tömböt, ezt követően két változóban eltároljuk a tömb két dimenziójához tartozó elemszámokat (nevezhetjük a sorok és oszlopok számának), majd egymásba ágyazott ciklusokkal előállítjuk az összes lehetséges index-kombinációt, és értéket adunk az elemeknek.

```
int[,] matrix = new int[11, 13];
int numRows = matrix.GetLength(0);    // numRows értéke 11 lesz
int numCols = matrix.GetLength(1);    // numCols értéke 13 lesz

for (int i = 0; i < numRows; i++)
{
    for (int j = 0; j < numCols; j++)
    {
        matrix[i, j] = i + j;
    }
}
```

Listák használata

Szemben a tömbökkel, amelyek méretét (elemszámát) a létrehozáskor meg kell adnunk, és azon később nincs lehetőségünk módosítani, a lista (`List`) egy dinamikus gyűjtemény, amely automatikusan növeli vagy csökkenti a méretét az aktuálisan benne lévő elemek számának megfelelően.

A tömbhöz hasonlóan a lista használatának első lépése is a lista változójának deklarációja, illetve a lista létrhozása (a szintaxis eltér a tömböknél megszokottól). Az alábbi első példában egy egészekből álló üres listát hozunk létre, a második példában három karakterlánccal inicializáljuk a létrehozott karakterláncokból álló listát.

```
List<int> listOfNumbers = new List<int>();
List<string> listOfStrings = new List<string> { "Trust", "No", "One" };
```

A listához további elemeket adhatunk az **Add** utasítással, a **Remove** utasítással pedig törörhetünk belőle egy megadott elemet (annak első előfordulását). Az aktuálisan a listában lévő elemek számát a `Count` utasítással, a teljes lista méretét (kapacitását) pedig a `Capacity` utasítással kérdezhetjük le. A tömbhöz hasonlóan a lista elemei elérhetők indexeken keresztül.

```
List<char> letters = new List<char>();
letters.Add('a');
letters.Add('b');
letters[1] = 'c';
for (int i = 0; i < letters.Count; i++)
{
    Console.Write(letters[i]);
}
```

A `foreach` ciklus

Különféle *gyűjtemények*, például tömbök és listák bejárására használható a `foreach` ciklus. Működésének lényege, hogy automatikusan kezeli a ciklus változójának léptetését, vagyis például egy tömb bejárásánál nem az elemek indexei alapján férünk hozzá azokhoz, hanem a `foreach` ciklus változója közvetlenül a gyűjtemény aktuális elemének értékét veszi fel minden lépésben.

A `foreach` ciklus a gyűjtemény elemeit sorban végigveszi, és minden iteráció során az adott elem értékét hozzárendeli egy ideiglenes változóhoz, amit a ciklusmagon belül használhatunk. Az iteráció addig folytatódik, amíg a gyűjtemény minden elemét be nem jártuk. Fontos megjegyzenünk, hogy a `foreach` ciklus épp ezért nem használható arra, hogy a gyűjtemény elemeit módosítsuk, mert a ciklus változója csak egy másolatot tartalmaz az adott elemről (ezért nem végezhetjük el a ciklusváltozó értékadását).

Az alábbi példában létrehozunk egy tíz elemű tömböt egész számokból, majd a tömböt bejárva számítjuk és kiírjuk az elemeinek négyzetét, amelyet hozzáadunk egy listához is.

```
int[] numbers = new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
List<int> squares = new List<int>();

foreach (int n in numbers)
{
    int squareNumber = n * n;
    Console.WriteLine($"{n}^2 = {squareNumber}");
    squares.Add(squareNumber);
}
```

Feladatok

1 Készítsünk programot, amely ciklusok használatával felsorolja a francia kártya lapjait egy tömbbe . A lehetséges színek: `Kőr`, `Káró`, `Treff` és `Pikk`. A lapoknak 13 féle magassága lehet: számok 2-től 10-ig, majd `Jumbó`, `Dáma`, `Király` és `Ász`.

Példa

Az 52 elemű tömb elemei tehát:

```
{ "Kőr 2", "Kőr 3", ..., "Kőr Király", "Kőr Ász", "Káró 2", "Káró 3", ...,
"Pikk Dáma", "Pikk Király", "Pikk Ász" }
```

2 Keverjük meg a korábban készített kártyapaklit a *Fisher–Yates* keveréssel. A módszer lényege, hogy a tömb elemein végighaladva mindegyikhez kiválaszt egy véletlen helyen lévő elemet a korábban még nem vizsgáltak közül, amelyeket utána megcserél. Az algoritmus pszeudokóddal az alábbi formában adható meg (1-alapú indexelést használva).

ciklus $i \leftarrow$ 1-től $(n-1)$-ig
  $j \leftarrow$ véletlen egész; $i \leq j \leq n$
  $x[i] \leftrightarrow x[j]$
ciklus vége

3 Kérjünk el a felhasználótól előre megadott darabszámú szót, amelyeket tároljunk el egy tömbben. Ezután kérjünk el a felhasználótól egy további szót, és válaszoljuk meg az alábbiakat.

- Benne van-e a gyűjteményben a megadott szó?
- Ha benne van, hol található először?

4 Módosítsuk az előző feladat megoldását úgy, hogy a felhasználótól bekért szavakat egy listában tároljuk el, és a bekérést a STOP kulcsszó megadásakor fejezzük be. Ha szükséges, módosítsuk a két előbbi lekérdezést is. Milyen hasonlóságokat és különbségeket tapasztalunk a tömbök és listák használatában?

5 Felmérést végzünk barátaink programozói ismereteiről. Kérjük el az adott személy nevét (`string`), életkorát (`int`) és hogy rendelkezik-e programozói tapasztalattal (`bool`). A neveket, életkorokat és tapasztalatokat tároljuk három külön listában, amelyeket az kapcsol össze, hogy egy adott indexen egy konkrét személy adatait találjuk. A bekérést egy üres név megadásáig folytassuk. Ezt követően határozzuk meg az alábbiakat.

- Mi az átlagéletkor a teljes adathalmazban? (Használjuk a `foreach` utasítást a bejáráshoz.)
- Mi az átlagéletkor a programozói tapasztalat nélküli személyek között?
- Hány éves a legidősebb, programozó tapasztalattal rendelkező személy és mi a neve?

6 Hozzunk létre egy $N \times M$-es kétdimenziós tömböt $(1 < N, M < 10)$, amit töltsünk fel véletlenszerűen 0 és 9 közötti értékekkel. Jelenítsük meg a képernyőn ennek a mátrixnak az elemeit. Állítsuk elő a mátrix transzponáltját[1], vagyis tükrözzük azt a főátlójára.

Példa

$$\begin{matrix}1&2&3\\4&5&6\\7&8&9\end{matrix} \quad \rightarrow \quad \begin{matrix}1&4&7\\2&5&8\\3&6&9\end{matrix} \qquad\qquad \begin{matrix}a&b&c&d\\e&f&g&h\\i&j&k&l\end{matrix} \quad \rightarrow \quad \begin{matrix}a&e&i\\b&f&j\\c&g&k\\d&h&l\end{matrix}$$

7 Egy horgászverseny fogási adatait egy F táblázatban (kétdimenziós tömbben) tároljuk. $F(i, j)$ azt jelenti, hogy az i-edik horgász a j-edik halfajtából hány darabot fogott.

- Generáljuk le véletlenszerűen a táblázat adatait.
- Jelenítsük meg formázottan a fogási adatokat a képernyőn.
- Adjuk meg, hogy a horgászok mennyit fogtak az egyes halfajtákból.
- Melyik horgász fogta a legtöbb halat összesen?
- Volt-e olyan horgász, aki egyetlen halat sem fogott?

8 Kérjünk el a felhasználótól egy N pozitív egész értéket, és adjuk hozzá egy listához első elemként. Vegyük a lista utoljára hozzáadott elemét, legyen ez K. Ha K páros, adjuk hozzá a listához K felét, ha páratlan, akkor $3K + 1$-et. Addig ismételjük az előbbieket, amíg 1-et nem kapunk eredményül[2].

Kövessük nyomon a kiszámított érték és a lista állapotának változását hibakereső (debug) módban. Próbáljuk meg hibakeresés közben módosítani az aktuálisan kiszámított értéket.

9 Az alábbi algoritmussal szeretnénk az x tömb elemeit fordított sorrendben megkapni. Használjuk a hibakereső üzemmódot a hibák felderítésére és javítására.

```
int[] x = { 1, 2, 3, 4, 5, 6, 7, 8};
for (int i = 0; i < x.Length; i++)
{
    int tmp = x[i];
    x[i] = x[x.Length - i - 1];
    x[x.Length - i] = tmp;
}
```

10 Töltsünk fel egy egydimenziós tömböt megadott számú véletlen értékkel, majd valósítsuk meg az alábbi műveleteket, majd oldjuk meg a feladatot listával is.

- Válogassuk ki a gyűjtemény minden második elemét egy új gyűjteménybe.
- Fordítsuk meg a gyűjtemény elemeinek sorrendjét.
- Rendezzük a lehető legkisebb négyzetes mátrixba a gyűjtemény elemeit (az esetlegesen üresen maradó értékek helyére nulla kerüljön).

[1] https://hu.wikipedia.org/wiki/M%C3%A1trix_(matematika)#Transzpon%C3%A1l%C3%A1s

[2] A Collatz-sejtés szerint akármilyen pozitív számmal is kezdünk, a végén mindig elérjük az 1-et.

11 Készítsünk algoritmust, amely egy $N \times M$-es mátrix elemeit az óramutató járásának megfelelően $K \times 90°$-kal „elforgatja”, ahol K egész szám. Két példát mutatunk a $K = 1$ esetre.

Példa

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix} \quad \rightarrow \quad \begin{matrix} 4 & 1 & 2 \\ 7 & 5 & 3 \\ 8 & 9 & 6 \end{matrix} \qquad \begin{matrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{matrix} \quad \rightarrow \quad \begin{matrix} 5 & 1 & 2 & 3 \\ 9 & 10 & 6 & 4 \\ 13 & 11 & 7 & 8 \\ 14 & 15 & 16 & 12 \end{matrix}$$

12 Készítsünk egy egyszerű labirintus játékot. Töltsünk fel egy kétdimenziós tömböt véletlenszerűen `true` és `false` értékekkel. Adjunk meg egy kezdő koordinátát (indexet), majd határozzuk meg, hogy onnan eljuthatunk-e bármilyen úton a jobb alsó sarokba mindig csak szomszédos `true` mezőkre lépve. Egy adott elem szomszédai alatt a tőle balra és jobbra, valamint felette és alatta lévő elemeket értjük. A feltételeknek eleget tevő út nem minden esetben létezik. Ugyanígy előfordulhat, hogy több megfelelő útvonal is található a labirintusban.

Példa

$x = 2$ és $y = 1$ esetén egy lehetséges út például az alábbi (a kezdő és végpontok sötéttel jelölve).

T	F	T	T	T	F	T	T	T	T
F	F	T	T	F	F	T	F	T	T
F	T	T	T	T	T	T	F	F	T
F	F	T	F	F	F	F	T	T	T