A dinamikus tömb (lista)

Azonos típusú adatokat tárolhatunk tömbben (array). A tömb hátránya, hogy méretét már a deklarálásakor meg kell adni, és bár ez később módosítható, de nehézkesen. Egy elem beillesztése vagy törlése az összes mögötte levő elem léptetését igényli, ami hosszú ideig tart.

A dinamikus tömb a fent említett hátrányokra nyújt hatékonyabb megoldást. A dinamikus tömb dinamikusan (futás közben) módosítható méretű tömb. A dinamikus tömbbe a többi elem léptetése nélkül illeszthetünk újabb elemeket, illetve törölhetünk már létezőket.

A továbbiakban **a .NET nyelvhasználatának megfelelően a dinamikus tömböt listának nevezzük.** Ne keverjük össze a láncolt lista adatszerkezettel! A hagyományos tömböt statikus tömbnek is nevezzük.

A dinamikus tömbök használata felgyorsítja a programírást, és lerövidíti a kódot. Általában hatékonyabb programot eredményez, mintha statikus tömböt alkalmaznánk, főleg az elemek törlésénél vagy új elemek beillesztésénél. Gyakran nem kell foglalkoznunk az aktuális elem indexelésével sem. Az új elemet a dinamikus tömb végéhez fűzzük hozzá.

Listák a C# -ban

Az **ArrayList** egy érdekes lista, melybe úgy tűnik, hogy **vegyes típusú elemeket tehetünk** (ez azért van, mert az ArrayList alaptípusa Object). *De! ha a listába vegyes típusú elemeket helyezünk el, akkor később, a lista elemek feldolgozása során komoly nehézségekbe fogunk ütközni, ezért a nem professzionális szintű programozók számára ezt nem ajánljuk!*

A **típusos lista** létrehozásához pedig a **List<típus>** alakot használjuk, mely a generikus List osztályból készít el listát, a típus helyébe be kell írni a konkrét típust, és csak ilyen típusú elemeket tárolhatunk benne.

A listák kezdetben üresek (0 elemszámúak). Az **.Add() metódussal** lehet új elemmel bővíteni őket. Az aktuális kapacitását a listának (hány elemnek foglalt helyet) a **.Capacity** tulajdonsággal lehet lekérdezni. Az aktuális elemszámot a .**Count** –al lehet lekérdezni.

Típusos lista létrehozása

```
List<int> L = new List<int>();
```

Természetesen nem csak int típusú listák, hanem bármilyen típusú lista (string, double, …)létrehozható.

Típusos lista feltöltése

Az ilyen listákhoz is az .Add() metódussal tudunk új elemet hozzáadni:

```
L.Add( 12 );
L.Add( 45 );
int v = 68;
L.add( v ); // a 68 érték hozzáadása
```

A listához hiába próbálunk nem int típusú értékek hozzáadni, az fordítás közbeni hibát fog okozni:

```
L.Add( ”Hello” );
L.Add( true );
L.Add( 12.45 );
```

Típusos lista elemeinek száma

```
L.Count;
```

Típusos lista kiolvasása I.

Tegyük fel, hogy egy L nevű List<int>-hez már hozzáadtunk elemeket (valamennyit), és természetesen tudjuk, hogy mindegyik hozzáadott elem int típusú volt, hiszen egy ilyen típusos listához másmilyet nem is lehet hozzáadni.

Ekkor nagyon gyakran **for ciklussal olvassuk ki és dolgozzuk fel az elemeket**. **A lista tényleges elemszáma a .Count tulajdonságával lekérdezhető**. **A lista elemekre a vektoroknál már megszokott módon az elem sorszámával hivatkozhatunk.** A lista elemeinek sorszámozása 0-val kezdődik, és .Count-1 –ig tart.

```
int osszeg = 0;
for(int i=0;i<L.Count;i++)
      osszeg = osszeg + L[i];
```

Vegyük észre, hogy az ArrayList-el szemben itt nincs szükség a típuskényszerítésre, mivel a fordítóprogram az L típusával (List<int>) ki tudja következtetni, hogy az L[i] (az L lista i. eleme) típusa int!

Típusos lista kiolvasása II.

Amennyiben a cél a lista feldolgozása (nem módosítása) abban az esetben leggyakrabban foreach ciklust használunk

```
int osszeg = 0;
foreach(int x in L)
osszeg = osszeg + x;
```

A foreach ciklus deklarációjában is kijelentjük, hogy az L lista szerintünk int-eket tartalmaz (int x). Ezt a fordító jelen esetben le is ellenőrzi, mivel ismeri az L lista alaptípusát a List<int>-ből. Gyakorlatilag nem tudunk hibázni!

Lista elemeinek törlése

```
L.Clear();
```

Tömb elemeinek átmásolása egy listába

```
int[] arr = new int[3];
arr[0] = 2;
arr[1] = 3;
arr[2] = 5;
List<int> L = new List<int>(arr);
```

Tömb hozzáfűzése a listához

```
L.AddRange(arr);
```

Listából tömb létrehozása

```
int[] arr=L.ToArray(); //ha L típusa int!
```

Adott elem törlése

```
L.Remove(n); //az n értékű elemet törli (ha megtalálja)
```

Adott indexű elem törlése

```
L.RemoveAt(i); //az i indexű elemet törli
```

Adott indextől n elem törlése

```
L.RemoveRange(i,n); //az i indexű elemtől kezdve töröl n db elemet
```

Elem beszúrása adott indexű helyre

```
L.Insert(i, n); //az i indexű helyre beszúrja az n elemet
```

Tartalmazza-e?

```
L.Contains(n); //true, vagy false értéket ad vissza
```

Elem indexe

```
L.IndexOf(n); //visszatérési értéke az n elem indexe, vagy -1, ha nem eleme a
              listának
```

Rendezés

```
L.Sort();
```