VEZÉRLÉSI SZERKEZETEK

Ciklusok (folytatás)

Iterációs/Foreach ciklus

Szintaxisa:
foreach (vált_típus változó in változó_amit_nézek)
{
Utasítások;
}
Például: string s=„betűk sokasága”;
foreach (ch betu IN s)
{
Console.WriteLine(„az alábbi {0}betűkből áll a szöveg”,betu)
}

Véletlen szám generálása

Szintaxisa:
Random r=new Random();
Valtozonev = r.Next(paraméterek);
Random objektumot használja
Next metódust (eljárás/parancs) alkalmazunk
Paraméterei lehetnek:
() egy nem negatív véletlen számot ….
(max_érték) max_értéknél kisebb egész pozitív …
(min, max) min és max érték közötti egész + …

Típusok

- Egyszerű: Egész számok, valós számok, karakter, string, logikai
- Összetett: tömbök, halmazok, rekordok (osztály), állományok

Tömbök

Tömb: Olyan adatszerkezet, amely azonos típusú elemekből áll, és az elemeire sorszámmal hivatkozhatunk.
Deklarálása: típus[] név; Pl. int[] tömb;
Hely lefoglalása az elemeknek:
név= new típus[elemszám]
Például tömb=new int[10];
Programban tehát pl: **int[] vektor = new int[10];**
Értékadás:
Elemenként: név[index]=érték;
Int[] vek= new int[] {2,4,67,23}; (deklaráció ÉS értékadás!)

Többdimenziós tömb

int[,] matrix = new int[3, 3];
Értékadással összekötött deklaráció:
int[,] matrix = new int[,]
{ {12, 23, 2},
{13, 67, 52},
{45, 55, 1}
};

LISTÁK

- Tulajdonképpen 1dimenziós tömb, DE mérete nem statikus, hanem dinamikusan változik (pl.hozzáad)
- System.Collections.Generic névtérben található, ami azt jelenti, hogy a program elejére be kell írni:
- Using System.Collections.Generic;
- Minden listaelemhez egyforma sebességgel tudunk hozzáférni
- A listák a vektorokhoz hasonlóan kezelendők

LISTÁK típusai

- **Típusos LISTA**
- **Vegyes típusú LISTA (TömbLista/SorLista)**
- **Láncolt Lista**
  - **Egyirányú láncolt lista**
  - **Kétirányú láncolt lista**

Típusos LISTA

- ***Létrehozása***:(Referenciatípus, kell a new)

List <típus> listaváltozónév = new List<típus>();

  - Minden eleme ugyanolyan típusú
  - List osztályba tartozik (class)
- Nem töltődik fel alapértelmezetten 0 értékkel (lsd tömbök)
- HIVATKOZNI ugyanúgy [] jellel kell.

Típusos Listán értelmezhető tulajdonságok/műveletek-.
Lista elemének értékadása: *Listaváltozónév.Add(érték);*
Példa:
Létrehozunk egy sor nevű listát és beleteszünk 24-t

```
List<int> sor = new List<in>();
sor.Add(24);
 int vmi = int.Parse(Console.ReadLine());
        sor.Add(vmi);
```

Hivatkozni rá: sor[0];
FOREACH nagyon hasznos
Console.WriteLine(sor[0]);

Lista elemeinek értékadása véletlen számmal

```
for (int i=0; i<20; i++)
{
        Listavalt_nev.Add(r.Next(0,100);
}
```

Lista elemeinek Kiíratása

```
foreach (int elem in listavalt_nev)
        {
        Console.WriteLine(elem+" , ");
        }
```

Listán értelmezhető tulajdonságok

Count	mérete	sor.Count
Add	hozzáadás	
Capacity	kapacitás (memóriában lefoglalt terület)	
Clear	törlés	
Insert	beszúrás (adott index mögé) Insert(index, érték)	
Remove	eltávolítja az adott értékű elemet, pl. Remove(10)	
RemoveAt	adott indexű elemet törli	
Sort	rendez	
BinarySearch	Keresés (objektum indexét adja vissza	

pl. ha a lista elemek között van a 46, akkor

Console.WriteLine(sor[sor.BinarySearch(46)]);

mivel indexet ad vissza index=sor.BinarySearch

Vegyes típusú LISTA (TömbLista/SorLista)

Létrehozása:(Referenciatípus, kell a new)

ArrayList listaváltozónév = new ArrayList();

Nem javasolt a használata (inkább rekordot érdemes használni)

Tömblisták/Sorlisták műveletei

Ugyanazok a műveletek értelmezhetők + pl. **GetType()**

Pl. Értékadás ugyanúgy vagy sorlista[2].GetType()

valt_nev.Add(érték)

Példa: Array.List sorlista = new Array.List();

sorlista.Add(289);

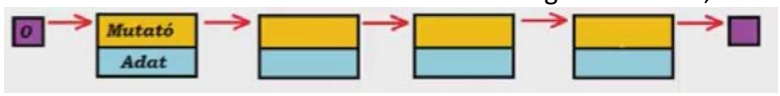
Listán belül még egy listát létre lehet hozni

Kiíratásnál FOREACH-ben **nem használhatok típust**!

```
foreach (var elem in sorlista)
        {
        Console.Write(elem+" , ");
        }
```

LÁNCOLT Listák

- A láncolt lista olyan adatszerkezet, amelynek elemei a soron következő elemre hivatkozó referenciát tartalmaznak
- A láncolt listát az első fej- vagy gyökérelemen keresztül érjük el.
- Ha az elemek csak a következő tagra mutatnak, akkor egyszeresen, láncolt listáról beszélünk (Egyirányú)



- Ha az elemek nem csak a következő tagra mutatnak, hanem a megelőző elemre is, akkor kétszeresen láncolt listáról beszélünk (Kétirányú)
- Csomópontokat értelmezünk, és a csomópontokban:
  - **Adatot**
  - **Mutatókat**

Láncolt Listán értelmezhető műveletek

Létrehozása

- LinkedList<típus> vált_név = new LinkedList<típus>();
- PL. **LinkedList<string> lancoltlista = new LinkedList<string>();**

Első helyre beszúr egy elemet

- AddFirst(); Pl. **lancoltlista.AddFirst("cola");**
- A First az első tagra mutat

Utolsó helyre beszúr egy elemet

- AddLast() Pl**. lancoltlista.AddLast("uccsó");**

A Last az utolsó tagra mutat

Csomópontok adatainak lekérdezése (Node)
LinkedListNode<string> aktualiselem = list.First;
Console.WriteLine("a cspont értéke: "+aktualiselem.Value);
Következő elemre ugrás NEXT, előzőre PREVIOUS metódussal
aktualiselem= aktualiselem.Next;

- Adott helyre való beszúrás (nem kell az összes többi elemet arrébb tenni, elég a mutatókat átírni)
- AddAfter(csomópont,érték)

Pl. Lancoltlista.AddAfter (aktualiselem,"beszúrandó");

Végignézni a láncolt_lista adatelemeit (utolsó elem a null elem)

```
while(aktualiselem != null)
{
Console.WriteLine(aktualiselem.Value);
aktualiselem = aktualiselem.Next;
}
```

Megjegyzés: Ha egyszer végigmentem a listán, akkor egy újabb feldolgozáshoz a mutatót állítani kell!

REKORDOK

Rekordok fogalma

- A rekord mezőkből épül fel, melyek mindegyike különböző típusú is lehet
- A mezők típusa általában valamilyen egyszerű típus (int, double, stb.)
- A mezőknek egyedi nevük van
- Nem beépített típus
  - STRUKTÚRAKÉNT hozom létre
  - CLASS-ként (osztályként) hozom létre

Rekordok létrehozása STRUKTÚRAKÉNT
Kulcsszó: STRUCT
Megadjuk a mezőneveket típusukkal együtt
PUBLIC kulcsszót mezőnév elé tesszük, használat miatt
Pl. 3 mezővel deklarált rekord (CSAK TÍPUS!!!)

```
struct DVD
{
public string Cime;
public int Ara;
public int MegjelenesEve;
}
```

Pl. film nevű változó létrehozása, mely rekord típusú
DVD film;

Rekord típusú változónak értékadása

A változó mezőire a **változónév.mezőnév** formában hivatkozhatunk
Pl.

```
film.Cime = "Underworld";
film.Ara = 5700;
film.MegjelenesEve = 2005;
```

Kiíratása:
Console.WriteLine(film.Cime);
Több változó egyszerre történő deklarálása: **DVD egyik, masik;**

Rekord alapú Tömbök értelmezése

Úgy képzeljük el, hogy a típusok helyébe kerül a rekord

	Név	Osztkód	Szül_Év	Átlag
1	Gazsi	9.e	2000	2,78
2				
3				
4				
5				
6				
7				
8				
9				
10				
11	Tódor	7.k	1999	3,11

Rekord alapú Tömbök Deklarálása

Előbbi példánál maradva:

struct rektipus
{
public string nev;
public string osztkod;
public int szul_ev;
public double atlag;
}

rektipus[] emberek=new rektipus[35];

Programban való hivatkozás

A logika ugyanaz, tehát index-szel hivatkozok a megfelelő vektor elemre, DE movel ez egy rekord, így még itt is tovább kell „szűkítenem”, azaz mezőnevet kell adnom

Pl.

emberek[2].atlag
emberek[4].szul_ev

Rekordok létrehozása CLASSKÉNT

Kulcsszó: CLASS

Mezőneveket típusukkal, PUBLIC kulcsszót mezőnév elé

Pl. 3 mezővel deklarált rekord (CSAK TÍPUS!!!)

class DVD
{
public string Cime;
public int Ara;
public int MegjelenesEve;
}

Pl. film nevű változó létrehozása, mely rekord típusú

DVD film;
film = new DVD();

Szokás kezdőértéket adni már a CLASS deklarációnál:

Pl.

class DVD
{
public string Cime = ”ismeretlen”;
public int Ara = 0;
public int MegjelenesEve= 1990;
}