.Net adatstruktúrák

Készítette:

Major Péter

Adatstruktúrák általában

- A .Net-ben számos nyelvvel ellentétben nem kell bajlódnunk a változó hosszúságú tömbök, listák, sorok stb. implementálásával, mert ezek megtalálhatók a függvénykönyvtárakban.
- A **System.Collection.Generic** névtér osztályai valósítják meg őket.
- Generikusak: bármely adattípusú változók tárolhatóak bennük.
- Ezeket érdemes használni, hiszen segítségükkel gyorsan érdekes programokat készíthetünk.
- Hátrányuk, hogy jóval lassabban működnek, mint egy célirányos implementáció, így teljesítmény centrikus esetben használatuk nem ajánlott.

A lista - List

- Gyakorlatilag olyan, mint egy változó hosszúságú tömb.
- Konstruktora paraméter nélküli, generikus:
  - **List**<*T*>(): adott adattípusú lista létrehozása, ahol *T* egy típus (pl.: *int*)
- Fő metódusai:
  - *void* **Add**(*T* item): elem hozzáadása
  - *void* **AddRange**(*IEnumerable*<*T*> collection): több elem hozzáadása egyszerre (pl.: tömb, másik lista)
  - *bool* **Remove**(*T* item): elem általvolítása, ha false-t ad vissza akkor az elem nem szerepel a listán
  - *void* **RemoveAt**(*int* index): adott indexű elem eltávolítása
  - *int* **IndexOf**(*T* item): elem sorszámát adja vissza (ha nincs a listán, akkor pedig -1-et)
  - *void* **Clear**(): a lista kiürítése
  - *T[]* **ToArray**(): a lista átalakítása tömbbé
  - *int* **Count** { get; }: elemek száma
- Elemek elérése a [] operátorral.

A lista - List

```
//Lista létrehozása
List<int> lista=new List<int>();

//Elemek hozzáadása
lista.Add(2);
lista.Add(3);
lista.Add(5);

//Összeg kiszámítása
int osszeg = 0;
for (int i = 0; i < lista.Count; i++)
{
    osszeg += lista[i];
}
Console.WriteLine(osszeg);

//Lista kiürítése
lista.Clear();
```

A sor - Queue

- A sor adatstruktúrát valósítja meg.
- Konstruktora generikus, paraméter nélküli:
  - **Queue**<*T*>(): sor létrehozása, *T* az elemek típusa (pl.: *int*)
- Fő metódusok:
  - *void* **Enqueue**(*T* item): elem hozzáadása a sorhoz
  - *T* **Dequeue**(): elem kiolvasása a sorból, ezután az elem törlődik a sorból (FIFO: first in first out, az elsőnek berakott elem kerül kiolvasásra legelőször)
  - *void* **Clear**(): a sor kiürítése
  - *T[]* **ToArray**(): a sor átalakítása tömbbé
  - *int* **Count** { get; }: elemek száma
- A sor belsejében lévő elemek nem olvashatóak ki.

A sor - Queue

```
//Sor létrehozása
Queue<int> sor = new Queue<int>();

//Elemek hozzáadása
sor.Enqueue(1);
sor.Enqueue(3);
sor.Enqueue(4);

//Összeg kiszámítása
int osszeg = 0;
while (sor.Count > 0)
{
    osszeg += sor.Dequeue();
}

Console.WriteLine(osszeg);
```

A verem - Stack

- A verem adatstruktúrát valósítja meg.
- Konstruktora generikus, paraméter nélküli:
  - **Stack**<*T*>(): verem létrehozása, *T* az elemek típusa (pl.: *int*)
- Fő metódusok:
  - *void* **Push**(*T* item): elem hozzáadása a veremhez
  - *T* **Pop**(): elem kiolvasása a veremből, ezután az elem törlődik a vermből (LIFO: last in first out, az utolsónak berakott elem kerül kiolvasásra legelőször)
  - *void* **Clear**(): a verem kiürítése
  - *T[]* **ToArray**(): a verem átalakítása tömbbé
  - *int* **Count** { get; }: elemek száma
- A verem belsejében lévő elemek nem olvashatóak ki.
- A verem működési szempontból a sortól csak az elemek kiolvasási sorrendjében tér el.

A verem - Stack

```
//Verem létrehozása
Stack<int> verem = new Stack<int>();

//Elemek hozzáadása
verem.Push(4);
verem.Push(6);
verem.Push(6);

//Összeg kiszámítása
int osszeg = 0;
while (verem.Count > 0)
{
    osszeg += verem.Pop();
}

Console.WriteLine(osszeg);
```

A szótár - Dictionary

- A szótár elempárok tárolására szolgál, melyek közül egyik a kulcs, amely azonosítja az elempárt, másik az érték, minden kulcs egyedi.
- Gyakorlatilag a szótár úgy viselkedik, mint egy lista, de az elemek indexe itt tetszőleges típusú lehet (pl.: szöveg).
- Konstruktora generikus, paraméter nélküli:
  - **Dictionary**<*TKey*, *TValue*>(): létrehoz egy szótárt, ahol *TKey* a kulcs, *TValue* az érték típusa
- Fő metódsai:
  - *void* **Add**(*TKey* key, *TValue* value): a value érték hozzáadása key kulccsal
  - *bool* **ContainsKey**(*TKey* key): megadja, hogy szerepel-e egy kulcs a szótárban
  - *bool* **ContainsValue**(*TValue* value): megadja, hogy szerepel-e egy érték a szótárban
  - bool **Remove**(*TKey* key): eltávolít egy elemet a szótárból, a visszatérési érték a művelet sikerességét jelzi
  - *int* **Count** { get; }: az elemek (kulcsok) száma
  - *void* **Clear**(): szótár kiürítése
- A szótár elemei a [] operátorral érhetőek el.
  - A szótárat foreach ciklussal lehet végig olvasni, amellyel a szótárból **KeyValuePair**<*TKey*, *TVaule*> típusú elemeket kapunk. Ezek **Key** és **Value** mezői adják a megfelelő kulcs és érték párokat.

A szótár - Dictionary

```
//Egy 20 elemű tömb előállítása véletlen számokkal 0..9 intervallumban
int[] szamok = new int[20];
Random R=new Random();
for (int i = 0; i < 20; i++)
{
    szamok[i] = R.Next(10);
}

//Szótár létrehozása
Dictionary<int, int> szamokSzama = new Dictionary<int, int>();

//Megszámoljuk, hogy hányszor fordul elő egy adott szám a tömbben
for (int i = 0; i < szamok.Length; i++)
{
    if (szamokSzama.ContainsKey(szamok[i]))
    {
        szamokSzama[szamok[i]]++;
    }
    else
    {
        szamokSzama.Add(szamok[i], 1);
    }
}

//Kiíratjuk melyik számból hány volt a tömbben
foreach(KeyValuePair<int,int> elem in szamokSzama)
{
    Console.WriteLine(elem.Key.ToString() + ": " + elem.Value + " db");
}
```

Elemek összehasonlítása

- Sok esetben lehet szükségünk az elemek sorba rendezésére, ehhez a **System.Collections.Generic**.*IComparer*<*T*> interfészt megvalósító osztályra van szükségünk. A *T* azon adattípus, amelyen majd összehasonlítást végzünk.
- Ennek fő metódusa:
  - *int* **Compare**(*T* x, *T* y): összehasonlít két azonos típusú elemet, visszatérési értéke:
    - negatív, ha x kisebb, mint y
    - nulla, ha x egyenlő y
    - pozitív, ha x nagyobb, mint y

Szótár elemek összehasonlítása

- Az összehasonlító osztály, amely *KeyValuePair<string, int>* típusú adatokat hasonlít össze, nagyság szerint csökkenő rendezéshez a **Value** szerint:

```
class WordComparer : IComparer<KeyValuePair<string, int>>
{
    public int Compare(KeyValuePair<string, int> x, KeyValuePair<string, int> y)
    {
        if (x.Value == y.Value) return 0;
        if (x.Value > y.Value) return -1;
        else return 1;
    }
}
```

- Használat:

```
WordComparer WC;
WC = new WordComparer();
List<KeyValuePair<string, int>> wordList =
    new List<KeyValuePair<string, int>>(Words.Count);
wordList.AddRange(Words);
wordList.Sort(WC);
```