Haladó fejlesztési technikák

XML formátum
JSON formátum
Adatstruktúra-független műveletek: LINQ
XLINQ

LINQ operátor példák

```
int[] first = new int[] { 2, 4, 6, 8, 2, 1, 2, 3 };
int[] second = new int[] { 1, 3, 5, 7, 1, 1, 2, 3 };
string[] strArray = new string[] { "Béla", "Jolán",
   "Bill", "Shakespeare", "Verne", "Jókai" };
List<Student> students = new List<Student>();
students.Add(new Student("Első Egon", 52));
students.Add(new Student("Második Miksa", 97));
students.Add(new Student("Harmadik Huba", 10));
students.Add(new Student("Negyedik Néró", 89));
students.Add(new Student("Ötödik Ödön", 69));
```

LINQ operátor példák – halmazok

- Két gyűjtemény egymás után fűzése (NEM halmazok!):

```
var allNumbers = first.Concat(second);
```

- Elem létezésének vizsgálata:

```
bool doesContainFour = first.Contains(4);
```

- Ismétlődések kivágása (halmazzá alakítás):

```
var onlyDifferentNumbers = first.Distinct();
```

- Halmazelméleti metszet:

```
var sameItems = first.Intersect(second);
```

- Halmazelméleti unió:

```
var unionOfSets = first.Union(second);
```

- Halmazelméleti különbség

```
var diffOfSets = first.Except(second);
```

LINQ operátor példák – sorrendezés

- OrderBy
  - Paraméterül egy olyan metódust (lambdát) vár, amely egy elemből meghatározza a kulcsot (azt az adatot, ami alapján rendezni fog – ez IComparable, vagy kell saját IComparer)
  - Az eredménye mindig IEnumerable<T>
  - Int tömb, rendezés a számok alapján:

```
var result = first.OrderBy(x => x);
```

  - String tömb, rendezés az elemek hossza alapján:

```
var result = strArray.OrderBy(x => x.Length);
```

  - Diákok listája, névsorba rendezés :

```
var result = students.OrderBy(x => x.Name);
```

  - Exception, mert nem Student : IComparable

```
var result = students.OrderBy(x => x);
```

LINQ operátor példák – szűrés, darabszám

- **Where / Count**
  - **A paraméterül adott lambda kifejezés eredménye bool**
  - **A *Where* eredménye az a gyűjtemény, ahol ez a lambda *true* értéket ad vissza**
  - **A *Count* eredménye a darabszám (int!), és meghívható paraméter nélkül is ➔ teljes darabszám**
  - **Int tömb, a páratlanok:**

```
var result = first.Where(x => x % 2 == 1);
```

  - **String tömb, a négy betűs nevek:**

```
int result= strArray.Count(x => x.Length == 4);
```

LINQ operátor példák – szűrés, részkiválasztás

- **Diákok listája, ahol a kreditszám prím:**

```
var result = students.Where(x =>
{
    if (x.Credits <= 1) return false;
    for (int i = 2; i <= Math.Sqrt(x.Credits); i++)
        { if (x.Credits % i == 0) return false; }
    return true;
});
// Második Miksa - 97, Negyedik Néró - 89
```

- **Tulajdonság kiválasztása / konverzió:**

```
var nameCollection = students.Select(x => x.Name);
var jsonCollection = students.Select(x => x.ToJson())
```

LINQ operátor példák – láncolás, lekérdezés

- **Diákok listája, a páratlan kreditszámúak nagybetűs neve név szerinti fordított sorrendben:**

```
var result= students.Where(x => x.Credits % 2 == 1)
   .OrderBy(x => x.Name)
   .Reverse()
   .Select(x => x.Name.ToUpper());
```

- **Ugyanaz az eredmény, ugyanaz a köztes kód, DEKLARATÍV megközelítésben:**

```
var result = from x in students
        where x.Credits % 2 == 1
        orderby x.Name descending
        select x.Name.ToUpper();
```

LINQ operátor példák – aggregálás

- **Aggregáló metódusok**

```
int totalSum = first.Sum(); //28
double averageOfItems = second.Average(); //2.875
int sumOfEvenItems = first
  .Where(x => x % 2 == 0).Sum(); //24
int sumOfOddItems = second
  .Where(x => x % 2 == 1).Sum(); //4
```

- **A fenti példa gyakori: valamilyen ismétlődés szerint akarom csoportosítani a gyűjteményemet, és az egyes csoportokra szeretném tudni a darabszámot/összeget**
  - Több hasonló utasítással oldható meg…
  - Ezen a Sum/Average Func<T, bool> paraméterezhetősége sem segít
- **Helyette inkább automata csoportosítás: GroupBy**

LINQ operátor példák – csoportosítás

- Csoportosítás, paritás szerinti darabszámok:

```
var groups = first.GroupBy(x => x % 2);
   // IEnumerable<IGrouping<TKey, TElement>>
foreach (var g in groups)
{
    Console.WriteLine("Remainder: " + g.Key +
        ", Number of items: " + g.Count());
}
```

- Ugyanez jobb a query syntax használatával

```
var result = from x in first
  group x by x % 2 into g
  select new {Remainder=g.Key, NumItems=g.Count()};
```

LINQ operátor példák – csatolás (inner join)

- Amennyiben van azonos adatmező két gyűjteményben, akkor van erre lehetőségünk (ld. Gyakorlat és Db-Linq és adatbázisok óra)
- Ez már tipikusan az, ami query syntax nélkül nagyon bonyolult (4 generikus típusparaméterrel rendelkező generikus metódus, két gyűjteménnyel és 3 lambda kifejezéssel…)
- ```
  var result = from firstItem in firstCollection
               join secondItem in secondCollection
               on firstItem.X equals secondItem.Y
  ```
- A query többi részében minden firstItem-hez a HOZZÁ ILLESZKEDŐ secondItem elem csatolódik, és mindkettő használható (pl. autóhoz a hozzá illeszkedő márka)
- ```
  var result = from car in carCollection
               join brand in brandCollection
               on car.brandId equals brand.Id
  ```

JSON Linq

```
// JObject obj = JObject.Parse(json);
// obj["property"]?.ToString();
JArray array = JArray.Parse(json);
Console.WriteLine(array[0].ToString());
Color firstColor = array[0].ToObject<Color>();
Console.WriteLine($"FIRST COLOR: {firstColor.Red} - {firstColor.Green}
  {firstColor.Blue}");

var q = from color in array.Children()
        group color by color["Red"] into grp
        orderby grp.Key
        select new { RedValue = grp.Key, PixelCount = grp.Count() };
foreach (var item in q) Console.WriteLine(item);
Console.ReadLine();
```

LINQ to XML, XLINQ

- **X* osztályok: igen erős LINQ-támogatás!**
  - **LINQ-zható IEnumerable<T>-ként kapunk vissza egy csomó adatot**

Pl. XElement xe2:

- **xe2.Descendants()**
  - **minden gyerekelem (gyerekelemek gyerekelemei is)**
- **xe2.Descendants("note")**
  - **ilyen nevű gyerekelemek (gyerekelemek gyerekei is)**
- **xe2.Elements()**
  - **közvetlen gyerekelemek**
- **xe2.Elements("note")**
  - **ilyen nevű közvetlen gyerekelemek**
- **xe2.Attributes(), xe2.Ancestors() …**

LINQ to XML

```
<people>
  <person id="43984">
    <name>Joe</name>
    <age>25</age>
    <phone>0618515133</phone>
  </person>
…
</people>
```

```
XDocument XDoc = ...
var q = from node in XDoc.Descendants("person")
        where node.Element("name").Value.StartsWith("J")
        select node;
```

Példa

http://users.nik.uni-obuda.hu/prog3/_data/people.xml

```
<person>
  <name>Dr. Vámossy Zoltán</name>
  <email>vamossy.zoltan@nik.uni-obuda.hu</email>
  <dept>Alkalmazott Informatikai Intézet</dept>
  <rank>egyetemi docens, intézetigazgató-helyettes, TDK-fel
  <phone>+36 (1) 666-5550</phone>
  <room>BA.3.12</room>
</person>
```

Példa

Listázzuk azokat, akik nem a BA épületben dolgoznak
(Alternatívák: közvetlenül XML-ből vagy köztes osztályt használva?)

```
XDocument XDoc = XDocument.Load("http://users.nik.uni-obuda.hu/prog3/_data/people.xml");
var q0 = from node in XDoc.Descendants("person")
                let room=node.Element("room").Value
                where !room.StartsWith("BA")
                select node.Element("name").Value;
foreach (var item in q0) {
      Console.WriteLine(item);
}
```

Példa

- **Köztes osztály használatával az XML node-ból először objektumot konvertálunk, így visszavezetjük az XML feldolgoztást Linq To Objects módszerre**

```
class Person
{
    public static Person Parse(XElement node)
    {
        return new Person()
        {
            Name = node.Element("name")?.Value,
            Email = node.Element("email")?.Value,
            Dept = node.Element("dept")?.Value,
```

```
// IEnumerable<X> vs List<X>
public static IEnumerable<Person> Load(string url)
{
    XDocument XDoc = XDocument.Load(url);
    return XDoc.Descendants("person").
        Select(node => Person.Parse(node));
}
```

Példa – Extension Method

```
static class MyExtensions
{
    public static void ToConsole<T>(
        this IEnumerable<T> input, string str)
    {
        Console.WriteLine("*** BEGIN " + str);
        foreach (T item in input)
        {
            Console.WriteLine(item.ToString());
        }
        Console.WriteLine("*** END " + str);
        Console.ReadLine();
    }

}
```

```
IEnumerable<Person> people = Person.
    Load("http://users.nik.uni-obuda.hu/prog3/_data/
      people.xml");
people.Select(person => person.Name).
    ToConsole("ALL WORKERS");
```

Példa – Dolgozók száma és oldalakra darabolt lista

```
string dept = "Alkalmazott Informatikai Intézet";
int num = people.
    Where(person => person.Dept == dept).
    Count();
int num2 = people.
    Count(person => person.Dept == dept);
```

```
int current = 0; int pagesize = 15;
while (current < num)
{
    var q2 = people.
        Where(person => person.Dept == dept).
        Skip(current).
        Take(pagesize).
        Select(person=>person.Name);
    q2.ToConsole("Q2 / page");
    current += pagesize;
}
```

Példa – Legrövidebb és leghosszabb nevek

```
// 3. people with the longest/shortest name
// Query vs Method syntax???
var q3 = from person in people
         let minlen = people.Min(x => x.Name.Length)
         let maxlen = people.Max(x => x.Name.Length)
         where person.Name.Length == minlen ||
                 person.Name.Length == maxlen
         select new { person.Name, person.Name.Length };
q3.ToConsole("Q3");
```

Példa – Csoportok; Legnagyobb csoport

```
// 4. number of people per department
var q4 = from person in people
         group person by person.Dept into g
         select new { Dept = g.Key, Cnt = g.Count() };
q4.ToConsole("Q4");
```

```
// 5. biggest dept
// ElementAt, First, Last, Single, ...OrDefault
var oneDept = q4.
    OrderByDescending(rec=>rec.Cnt).
    FirstOrDefault();
var oneDept_alter = q4.
    Aggregate((i, j) => i.Cnt > j.Cnt ? i : j);
Console.WriteLine(oneDept.ToString());
Console.WriteLine(oneDept_alter.ToString());
```

Feladat

http://users.nik.uni-obuda.hu/prog3/_data/war_of_westeros.xml

```
<battle>
  <name>Battle of the Golden Tooth</name>
  <year>298</year>
  <outcome>attacker</outcome>
  <type>pitched battle</type>
  <majordeath>1</majordeath>
  <majorcapture>0</majorcapture>
  <season>summer</season>
  <location>Golden Tooth
  <region>The Westerlan
  <attacker>...</attack
  <defender>...</defend
</battle>
```

```
<defender>
  <king>Robb Stark</king>
  <commanders>
    <commander>Clement Piper</commander>
    <commander>Vance</commander>
  </commanders>
  <house>Tully</house>
  <size>4000</size>
</defender>
```

Feladat

Az öt király háborújában …

1. **Hány ház vett részt?**
2. **Listázzuk az „ambush” típusú csatákat!**
3. **Hány olyan csata volt, ahol a védekező sereg győzött, és volt híres fogoly?**
4. **Hány csatát nyert a Stark ház?**
5. **Mely csatákban vett részt több, mint 2 ház?**
6. **Melyik volt a 3 leggyakrabban előforduló régió?**
7. **Melyik volt a leggyakoribb régió?**
8. **A 3 leggyakrabban előforduló régióban mely csatákban vett részt több, mint 2 ház? (Q5 join Q6)**
9. **Listázzuk a házakat nyert csaták szerinti csökkenő sorrendben!**
10. **Mely csatában vett részt a legnagyobb ismert sereg?**
11. **Listázzuk a 3 leggyakrabban támadó parancsnokot!**

Források

- Lambda expressions: http://msdn.microsoft.com/en-us/library/bb397687.aspx
- Lambda expressions: http://geekswithblogs.net/michelotti/archive/2007/08/15/114702.aspx
- Why use Lambda expressions: http://stackoverflow.com/questions/167343/c-lambda-expression-why-should-i-use-this
- Recursive lambda expressions: http://blogs.msdn.com/b/madst/archive/2007/05/11/recursive-lambda-expressions.aspx
- Standard query operators: http://msdn.microsoft.com/en-us/library/bb738551.aspx
- Linq introduction: http://msdn.microsoft.com/library/bb308959.aspx
- 101 Linq samples: http://msdn.microsoft.com/en-us/vcsharp/aa336746
- Lambda: Reiter István: C# jegyzet (http://devportal.hu/content/CSharpjegyzet.aspx) , 186-187. oldal
- Linq: Reiter István: C# jegyzet (http://devportal.hu/content/CSharpjegyzet.aspx) , 250-269. oldal

- Fülöp Dávid XLinq prezentációja
- Linq to XML in 5 minutes: http://www.hookedonlinq.com/LINQtoXML5MinuteOverview.ashx
- Access XML data using Linq: http://www.techrepublic.com/blog/programming-and-development/access-xml-data-using-linq-to-xml/594
- Simple XML parsing examples: http://omegacoder.com/?p=254 , http://gnaresh.wordpress.com/2010/04/08/linq-using-xdocument/
- XML: Reiter István: C# jegyzet (http://devportal.hu/content/CSharpjegyzet.aspx) , 224. oldal (A könyv az XMLReader/Writer, illetve az XmlDocument használatát mutatja be)