Tallinna Ülikool
Informaatika Instituut

LINQ andmehaldus

Jaagup Kippar

Tallinn 2011

Sisukord

Alustus

Language INtegrated Query nime all on .NET raamistikku ning sealtkaudu ka C# keelde kaasa tulnud meeldiv ja kasulik komplekt andmetega ümber käimise vahendeid. Esimese nähtava abilisena neist on lisandunud massiividele hulk käsklusi andmete lihtsamaks töötlemiseks. Lühikesed käsud, mis siiani olid nt. Ruby ja Pythoni-nimeliste keelte pärusmaa leiavad selle kaudu tee ka C# juurde. Teine hea omadus uuel tehnoloogial on, et sarnaselt on võimalik andmeid küsida väga erinevate andmeallikate juurest - massiivist andmebaasini. Juurde on loodud lihtsalt liides nimega IEnumerable mis tähistab igasugust järjestatavat samatüübilist andmehulka ning LINQ-ga seotud käske saab rakendada kõigile selle liidese reegleid järgivatele andmetele.

Esimesed käsklused

Et massiivid võimalik teha väikesed ja kergesti ligipääsetavad, siis alustame tutvustusnäiteid sealt.

Näitetulemuste mugavamaks väljatrükiks on kasutatud string-klassi Join-käsklust, mis loetelu

elemendid üheks tekstiks ühendab. Nii et koodilõigu

```
int[] m = {2, 6, 4, 5, 3, 2};
Console.WriteLine(string.Join(" ", m));
```

tulemuseks ekraanil on rida

```
2 6 4 5 3 2
```

Erinevate väärtuste kuvamiseks on käsklus Distinct

```
            IEnumerable<int> erinevad = m.Distinct();
```

mis siis tulemusena annab IEnumerable tüüpi muutujasse saadud erinevad tulemused. Tahtes loetelu tagasi massiiviks saada, aitab käsklus ToArray. Siis võimalik elementide poole taas otse järjekorranumbri kaudu pöörduda.

```
            int[] m2 = erinevad.ToArray<int>();
```

Trükkides uus massiiv välja on näha, et algsete andmetega võrreldes on sealt korduv 2 läinud kaduma.

```
2 6 4 5 3
```

Kõik see on võimalik ka ühe käsuga kirja panna

```
            int[] m3 = m.Distinct().ToArray();
```

Esimene näitlik koodifail tervikuna.

```
using System;
using System.Collections.Generic;
using System.Linq;

class Massiivitehted {
    public static void Main(string[] arg) {
            int[] m = {2, 6, 4, 5, 3, 2};
            Console.WriteLine(string.Join(" ", m));
            IEnumerable<int> erinevad = m.Distinct();
            Console.WriteLine(string.Join(" ", erinevad));
            int[] m2 = erinevad.ToArray<int>();
            Console.WriteLine(string.Join(" ", m2));
            int[] m3 = m.Distinct().ToArray();
            Console.WriteLine(string.Join(" ", m3));
    }
}

/*
2 6 4 5 3 2
2 6 4 5 3
2 6 4 5 3
2 6 4 5 3
*/
```

Järgnevalt veel mõned kasulikud käsklused. Elementide arvu loendis annab Count

```
        Console.WriteLine("Elementide arv: " + m.Count());
```

Tagurpidi järjekorras tulevad väärtused käsuga Reverse.

```
        Console.WriteLine("Tagurpidi: " + string.Join(" ", m.Reverse()));
```

Loetelu algusest soovitud arvu väärtuste välja võtmiseks sobib Take

```
        Console.WriteLine("Neli esimest: " + string.Join(" ", m.Take(4)));
```

Tahtes nende käskude abil lõpust soovitud kogust elemente võtta, on üheks võimaluseks käske kombineerida. Kõigepealt pöörata rida ümber, siis võtta kolm esimest ning tulemus taas ümber pöörata. Et kõik need käsud on seotud IEnumerable liidesega, siis saab eelmise käsu väljundi ilusti uue käsu sisendiks lugeda.

```
        Console.WriteLine("Kolm viimast: " + string.Join(" ",
              m.Reverse().Take(3).Reverse()));
```

Olemas ka traditsioonilised andmekogumitega seotud käsklused, mida võimalik sarnaselt käivitada. Suurima, vähima, summa ja keskmise leidmiseks lihtsalt vastavad inglisekeelsed sõnad juhul, kui kogumist on võimalik soovitut välja arvutada.

```
        Console.WriteLine("Suurim: " + m.Max());
        Console.WriteLine("Vähim: " + m.Min());
        Console.WriteLine("Summa: " + m.Sum());
        Console.WriteLine("Keskmine: " + m.Average());
```

Sobiva väärtuse kindlaks tegemiseks käsklus Contains

```
        Console.WriteLine("Sisaldab 3: " + m.Contains(3));
```

Ning näite kood tervikuna:

```
using System;
using System.Collections.Generic;
using System.Linq;

class Massiivitehted2{
    public static void Main(string[] arg) {
        int[] m = { 2, 6, 4, 5, 3, 2 };
        Console.WriteLine("Algne: " + string.Join(" ", m));
        Console.WriteLine("Elementide arv: " + m.Count());
        Console.WriteLine("Tagurpidi: " + string.Join(" ", m.Reverse()));
        Console.WriteLine("Neli esimest: " + string.Join(" ", m.Take(4)));
        Console.WriteLine("Kolm viimast: " + string.Join(" ",
              m.Reverse().Take(3).Reverse()));

        Console.WriteLine("Suurim: " + m.Max());
        Console.WriteLine("Vähim: " + m.Min());
        Console.WriteLine("Summa: " + m.Sum());
        Console.WriteLine("Keskmine: " + m.Average());

        Console.WriteLine("Sisaldab 3: " + m.Contains(3));
    }
}

/*
Algne: 2 6 4 5 3 2
Elementide arv: 6
Tagurpidi: 2 3 5 4 6 2
```

```
Neli esimest: 2 6 4 5
Kolm viimast: 5 3 2
Suurim: 6
Vähim: 2
Summa: 22
Keskmine: 3,66666666666667
Sisaldab 3: True
*/
```

Ülesandeid

* Loo massiiv kümnekonna täisarvuga
* Trüki välja neist suurim
* Trüki välja nelja esimese arvu summa
* Eralda neli esimest arvu uude massiivi
* Trüki loodud massiiv välja tagurpidises järjekorras

Hulgatehted

Kahe kogumi võrdlemiseks on ka mugavad käsud olemas. Union liidab kokku kahe kogumi väärtused eemaldades korduvad elemendid. Intersect leiab väärtused mis mõlemis olemas on. Except annab hulkade vahe, ehk näitab tulemusena vaid neid, mis esimeses hulgas on, aga teises ei ole. Kahe kogumi väärtuste lihtsaks kokkuliitmiseks sobib Concat.

```
using System;
using System.Collections.Generic;
using System.Linq;

class Massiivitehted3
{
    public static void Main(string[] arg)
    {
        string[] jukukeeled = { "eesti", "vene" };
        string[] katikeeled = { "eesti", "soome", "inglise" };
        Console.WriteLine("Kahe peale oskavad: " +
             string.Join(" ",jukukeeled.Union(katikeeled)));
        Console.WriteLine("Mõlemad oskavad (ühisosa): " +
             string.Join(" ", jukukeeled.Intersect(katikeeled)));
        Console.WriteLine("Ainult Juku oskab: " +
             string.Join(" ", jukukeeled.Except(katikeeled)));
        Console.WriteLine("Kõik oskused loetelus: " +
             string.Join(" ", jukukeeled.Concat(katikeeled)));

    }
}

/*
Kahe peale oskavad: eesti vene soome inglise
Molemad oskavad (ühisosa): eesti
Ainult Juku oskab: vene
Koik oskused loetelus: eesti vene eesti soome inglise
*/
```

Ülesandeid

* Loo kolm massiivi täisarvudega, mis tähistavad vastavas peatuses peatuvate bussiliinide numbreid
* Leia, kas leidub mõni bussiliin, mis peatuks kõigis kolmes peatuses
* Leia kõik bussiliinid, millega ei saa sõita esimesest peatusest teise
* Väljasta kõik erinevad liininumbrid

Avaldistega käsud

Sobivate andmete leidmiseks saab kasutada funktsioonide parameetrisse edastatavaid lambda-avaldisi. Massiivi

```
int[] m = { 2, 6, 4, 5, 3, 2 };
```

neljast suuremate väärtuste leidmiseks sobib käsklus

```
Console.WriteLine("Neljast suuremaid: " + m.Count(a => a > 4));
```

mis annab vastuseks

```
Neljast suuremaid: 2
```

Andmeid üle kontrollides paistab see õige olevat - neljast suuremad väärtused on 5 ja 6.

Avaldise a => a > 4 vasakul pool (enne => märki) on lihtsalt täht a. Siinsete käskude puhul tähendab see, et kogumi iga liikme puhul pannakse ta kõigepealt muutujasse nimega a ning seejärel tehakse temaga toimetus, mis kirjas avaldise paremal pool - kus võib vastavat muutujat kasutada. Count-funktsioon loendab kokku vaid need elemendid, kus avaldise puhul väljastatakse tõene tulemus, st. millise konkreetse elemendi puhul avaldis a>4 on tõene.

Kui tahta tingimusele vastavaid elemente näha ka, siis, selleks sobib käsklus Where. Tulemusena tagastatakse 6 ja 5.

```
Console.WriteLine("Neljast suuremad: "+
      string.Join(" ", m.Where(a => a > 4)));
```

Kui korraga vaja vaid ühte tingimusele vastavat väärtust, siis selle väljastab käsklus First.

```
Console.WriteLine("Esimene neist: " + m.First(a => a > 4));
```

Kogu massiivitäit andmeid mõne tehte abil muuta lubab Select. Siin näites jäävad täisarvud täisarvudeks, kuid vajadusel saab .NET neljandast versioonist alates väljastada tulemuse ka muu vajaliku andmetüübina. Näiteks on nõnda võimalik ühe käsuga muuta terve massiivitäis arve tekstideks või vastupidi.

```
Console.WriteLine("Kahega korrutatult: " +
     string.Join(" ", m.Select(a => a * 2)));
```

Ka elementide arv ei pea massiivi läbiva käsu puhul samaks jääma. SelectMany arvestab, et igast elemendist tekib vastusena loend ning kõikide nende loendite elementide väärtused korjatakse lõpuks ühte suurde jadasse kokku. Siin näites siis luuakse igast ettevõetavast arvust

kaheelemendiline massiiv, kus esimeseks elemendiks on ettevõetud arv ise ning teiseks tema ruut.

```
a => new int[]{a, a * a}
```

SelectMany korjab nüüd igast m-nimelises massiivi elemendist tekkinud arvupaarid kokku ning annab välja pika arvude ning ruutude jada.

```
          Console.WriteLine("Arvud ja ruudud: " +
            string.Join(" ", m.SelectMany(a => new int[]{a, a * a})));
```

Nõnda siis massiiv

```
int[] m = { 2, 6, 4, 5, 3, 2 };
```

annab eelpooltoodud SelectMany ning avaldise tulemusena välja jada

```
Arvud ja ruudud: 2 4 6 36 4 16 5 25 3 9 2 4
```

Loendist võimalik ka muul moel väärtusi kätte saada. Algusest kuni soovitud elemendini massiivis väljastab ahela käsklus TakeWhile - algusest võetakse väärtusi senikaua kuni avaldises olev tingimus pole täidetud. Ehk siis massiivist

```
        int[] m = { 2, 6, 4, 5, 3, 2 };
```

avaldis

```
                Console.WriteLine("Kuni väärtuseni 5: " +
            string.Join(" ", m.TakeWhile(a => a != 5)));
```

annab tulemuseks

```
Kuni väärtuseni 5: 2 6 4
```

Soovitud kohast edasi liikuda on võimalik käsuga SkipWhile

```
        Console.WriteLine("Alates väärtusest 5: " +
             string.Join(" ", m.SkipWhile(a => a != 5)));
```

annab sama massiivi puhul siis tulemuseks

```
Alates väärtusest 5: 5 3 2
```

```
using System;
using System.Collections.Generic;
using System.Linq;

class Massiivitehted4{
    public static void Main(string[] arg) {
        int[] m = { 2, 6, 4, 5, 3, 2 };
        Console.WriteLine("Neljast suuremaid: " + m.Count(a => a > 4));
        Console.WriteLine("Neljast suuremad: "+
             string.Join(" ", m.Where(a => a > 4)));
        Console.WriteLine("Esimene neist: " + m.First(a => a > 4));
```

```
        Console.WriteLine("Kahega korrutatult: " +
             string.Join(" ", m.Select(a => a * 2)));
           Console.WriteLine("Arvud ja ruudud: " +
             string.Join(" ", m.SelectMany(a => new int[]{a, a * a})));

                    Console.WriteLine("Kuni väärtuseni 5: " +
             string.Join(" ", m.TakeWhile(a => a != 5)));
        Console.WriteLine("Alates väärtusest 5: " +
             string.Join(" ", m.SkipWhile(a => a != 5)));
        Console.WriteLine("Kahanevalt sorditult "+
               string.Join(" ",m.OrderBy(a => -a)));
    }
}

/*
Neljast suuremaid: 2
Neljast suuremad: 6 5
Esimene neist: 6
Kahega korrutatult: 4 12 8 10 6 4
Arvud ja ruudud: 2 4 6 36 4 16 5 25 3 9 2 4
Kuni väärtuseni 5: 2 6 4
Alates väärtusest 5: 5 3 2
Kahanevalt sorditult 6 5 4 3 2 2
*/
```

Ülesandeid

* Loo massiiv inimeste pikkustega
* Loe kokku, mitu neist on pikemad kui 180 cm.
* Loo uus massiiv, kus inimesed on pikkuste järjekorras
* Trüki pikkused kuni esimese 180-cm pikkuse tegelaseni
* Trüki pikkused pärast esimest 180-cm pikkust tegelast
* Loo massiiv, kus kirjas on pikkused meetrites (nt. 1.65).
* Loo massiiv, kus pikkused kirjas tekstina (nt. 1m 65cm).

LINQ ja tekstifailid

Tekstifaili mugavaks lugemiseks ja kirjutamiseks on klassi System.IO.File juurde loodud mugavad käsklused ReadAllLines ning WriteAllLines. Nõnda saab kergesti lugeda failitäis teksti ridade kaupa massiivi või siis jällegi kirjutada sealt faili. Näitena loetakse mällu failis ujujad.txt olevad nimed. Nendest võetakse maha failis jooksjad.txt olevad nimed ning tulemus kirjutatakse faili ainultujujad.txt.

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;

class Failivordlus
{
    public static void Main(string[] arg)
    {
        File.WriteAllLines("ainultujujad.txt",
          File.ReadAllLines("ujujad.txt").Except(File.ReadAllLines("jooksjad.txt"))
        );
    }
}
/*
```

```
>type ujujad.txt
Kati
Siiri
Madis
Mart

>type jooksjad.txt
Juku
Kati
Mati

>type ainultujujad.txt
Siiri
Madis
Mart

>c:\WINDOWS\Microsoft.NET\Framework\v4.0.30319\csc.exe Failivordlus.cs
*/
```

Sarnane toimetusnäide ka arvudega. Sisendiks olevas tekstifailis on viie suusahüppekohtuniku antud punktid võistlejale. Vastavalt reeglitele eemaldatakse sealt suurim ja vähim väärtus ning ülejäänud kolmest arvutatakse keskmine. Selleks siis loetakse failis olevad read massiivi, Select käsu ja double.Parse abil muudetakse arvudeks. Järjestatakse arvu enese väärtuse järgi. Esimene jäetakse välja ning sealt alates jäetakse alles kolm elementi, mis järjestatuse tõttu on siis allesjääjad. Ning Average arvutab neist keskmise. (16,3+16,2+14,3)=15,6 - sama näitab ka programmi töö tulemus.

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;

class Failipunktid
{
    public static void Main(string[] arg)
    {
          Console.WriteLine(File.ReadAllLines("punktid.txt").Select(p => double.Parse(p))
                .OrderBy(p => p).Skip(1).Take(3).Average());
    }
}
/*
E:\jaagup\11\09\linq>type punktid.txt
16,3
16,2
17,4
12,5
14,3

E:\jaagup\11\09\linq>Failipunktid
15,6
*/
```

Ülesandeid

* Loo fail inimeste pikkustega
* Leia suurim pikkus
* Leia kõik keskmisest väiksemad pikkused
* Leia mediaan, st pikkus, millest on pooled suuremad ja pooled väiksemad

* Paiguta pikkused sorditуna teise faili
* Trüki teise faili kõigepealt paarisarvulised ning siis paarituarvulised pikkused.

Funktsioonid

Eelnevalt toodud näidetes pandi avaldiste juures tehtavad kontrollid ja tehted otse sulgude sisse nagu näiteks m.Count(a => a > 4). Sellist kirjapilti nimetatkse lambda-avaldiseks. Iseenesest on see lühendkuju olukorrast, kus käsklusele Count antakse ette funktsioon, mis saab parameetriks muutuja nimega a. Ning funktsioon väljastab avaldise a>4 väärtuse. Pikemate arvutuste korral aga ongi mõistlik arvutused päris eraldi funktsiooni paigutada. Siin on eraldi pisikese näitena toodud stringi pikkuse väljastav funktsioon pikkus.

```
    public static int pikkus(string tekst){
      return tekst.Length;
    }
```

Hiljem käivitamiseks piisab funktsiooni nimi lihtsalt avaldise sulgudesse panna.

```
        IEnumerable<int> pikkused=eesnimed.Select(pikkus);
```

Tulemusega saab ringi käia nagu tavalise IEnumerable-tüüpi andmestikuga.

C# juures on ka mugav foreach-nimeline käsklus kogumite andmete läbi käimiseks. Pole vaja teada elementide järjekorranumbreid, lihtsalt ükshaaval võetakse kogumi liikmed ette ning nendega on ploki sees võimalik sooritada soovitud toiming.

```
        foreach(String element in vastus){
          Console.WriteLine(element);
        }
```

Näite kood tervikuna:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

class Funktsioonid{
    public static string Sorenda(string tekst) {
      return string.Join(" ", tekst.ToCharArray());
    }
    public static int pikkus(string tekst){
      return tekst.Length;
    }
    public static void Main(string[] arg) {
        string[] eesnimed = { "Juku", "Kati", "Katrin", "Madis" };
         IEnumerable<int> pikkused=eesnimed.Select(pikkus);
         Console.WriteLine(String.Join(" ", pikkused));
        IEnumerable<string> vastus=eesnimed.Select(Sorenda);
        foreach(String element in vastus){
          Console.WriteLine(element);
        }
    }
}

/*
E:\jaagup\11\09\linq>Funktsioonid
```

```
4 4 6 5
J u k u
K a t i
K a t r i n
M a d i s
*/
```

Ülesandeid

* Loo massiivi inimeste enimedega

* Koosta funktsioon toimetuseks, kus nime eestäht on viidud lõppu.

* Töötle nõnda kõik nimed Select-käsu abil.

* Koosta funktsioon nimede tähtede tähekoodide kokkuliitmiseks.

* Leia nõnda igale massiivis olevale sõnale vastav räsikood.

* Koosta funktsioon, kus etteantud sõna põhjal luuakse massiiv, kus esimeseks liikmeks on sõna esimene täht, teiseks kaks esimest jne. kuni viimase liikme juures on tegemist terve sõnaga. Kogu SelectMany abil nõnda kõikidest algse massiivi sõnalõikudest tekkivad jupid kokku ühte suurde massiivi.

Klassid

Objektorienteeritud programmeerimises grupeeritakse andmed ja funktsioonid klassidesse. Ning LINQ võimaldab ka sellisel kujul andmestikega mugavalt toimetada.

Siin näitena võetud lihtsalt üks kahemõõtmelise tasandi punkt oma x ja y koordinaadiga. Juures funktsioonid väärtuste küsimiseks ükshaaval ning terviktekstina.

```
class TasandiPunkt {
    int x, y;
    public TasandiPunkt(int ux, int uy) {
        x = ux; y = uy;
    }
    public int KysiX() { return x; }
    public int KysiY() { return y; }
    public override String ToString() { return "(" + x + " " + y + ")"; }
}
```

Kõik kolmest suurema x-iga punktid saab Where-käskluse abil kätte järgnevalt:

```
IEnumerable<TasandiPunkt> vastus2 =  punktid.Where(p => p.KysiX()>3);
```

Sama tulemuse annab SQL-i laadse LINQ-süntaksiga päring:

```
IEnumerable<TasandiPunkt> vastus =
            from p in punktid where p.KysiX() > 3 select p;
```

Pikema päringu puhul on lähenemised sarnasemadki. Järgnevas näites küsitakse kolmest suuremate x-idega punktide y-koordinaadid.

```
Console.WriteLine(string.Join(" ",
            from p in punktid where p.KysiX() > 3 select p.KysiY()));

Console.WriteLine(string.Join(" ",
```

```
                    punktid.Where(p => p.KysiX()>3).Select(p=>p.KysiY())));
```

Liides IEnumerable on mõnes mõttes LINQ-toimingute aluseks. See on küljes massiivil, listil ning mõnel muulgi andmekogumit sisaldaval klassil. Vajadusel aga saab seda tüüpi väljundit ka ise luua. Lisand yield returni juures annab teada, et funktsiooni töö jooksul yield-i abil antud väärtused kogutakse kokku ning tagastatakse tervikuna IEnumerable-tüübile vastava komplektina. Siin näites luuakse nõnda klassi eksemplarile ette antud koguses arvude ruute.

```
    class RuutudeLooja {
        int kogus;
        public RuutudeLooja(int uusKogus) {
            kogus = uusKogus;
        }
        public IEnumerable<int> Ruudud() {
            for (int i = 1; i <= kogus; i++) {
                yield return i * i;
            }
        }
    }
```

Töö tulemusena näeb siis mitmel moel küsitud punktide andmeid. Samuti arvude ruute filtreerituna etteantud tingimuse järgi.

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;


namespace linq2 {
    class TasandiPunkt {
        int x, y;
        public TasandiPunkt(int ux, int uy) {
            x = ux; y = uy;
        }
        public int KysiX() { return x; }
        public int KysiY() { return y; }
        public override String ToString() { return "(" + x + " " + y + ")"; }
    }
    class RuutudeLooja {
        int kogus;
        public RuutudeLooja(int uusKogus) {
            kogus = uusKogus;
        }
        public IEnumerable<int> Ruudud() {
            for (int i = 1; i <= kogus; i++) {
                yield return i * i;
            }
        }
    }
    class Klassid {
        public static void Main(string[] arg) {
            TasandiPunkt[] punktid = {
                new TasandiPunkt(3, 5),
                new TasandiPunkt(4, 6)
             };
            IEnumerable<TasandiPunkt> vastus =
                        from p in punktid where p.KysiX() > 3 select p;
            Console.WriteLine(string.Join(" ", vastus));
            IEnumerable<TasandiPunkt> vastus2 =  punktid.Where(p => p.KysiX()>3);
            Console.WriteLine(string.Join(" ", vastus2));
            Console.WriteLine(string.Join(" ",
```

```
                    from p in punktid where p.KysiX() > 3 select p.KysiY()));
            Console.WriteLine(string.Join(" ",
                    punktid.Where(p => p.KysiX()>3).Select(p=>p.KysiY())));

                RuutudeLooja r = new RuutudeLooja(5);
            Console.WriteLine(string.Join(" ", r.Ruudud().Where(arv => arv > 10)));
        }
    }
}
/*
E:\jaagup\11\09\linq>Klassid
(4 6)
(4 6)
6
6
16 25
*/
```

Ülesandeid

* Loo klass mille väljadeks inimese eesnimi ja pikkus. Lisa konstruktor andmete sisestamiseks ning meetodid väärtuste küsimiseks.
* Koosta inimestest massiiv.
* Trüki välja kõikide eesnimed. Kasuta tulemuse saamiseks nii valemi (Select(...)) kui lause (from ...) kuju.
* Näita ainult nende eesnimesid, kelle pikkus on väiksem kui 180.
* Järjesta inimesed nende pikkuste järgi.

* Loo klass, mille eksemplar suudab genereerida soovitud koguse juhuarve etteantud vahemikus IEnumerable kujul.
* Meetodi parameetrina on võimalik määrata, kas arvude jaotus on ühtlane või on keskmisi väärtusi rohkem.

Grupeerimine

Grupeerimise tulemusena saab koondada sarnaste tunnustega andmed grupiti kokku - olgu siis inimesed nende elukohalinna või arvud mingite tunnuste alusel. Edasi võimalik siis juba gruppe lähemalt vaadelda - olgu elementide arvu loendada, keskmisi võtta või muul moel. Andmete loomiseks kasutame siin Enumerable.Range käsklust, saab ühekorraga valmis mitukümmend arvu.

Näitena jaotame nad gruppidesse vastavalt sellele, milline jääk kolmega jagamisel igast arvust jääb.

```
var grupid=from arv in arvud group arv by arv%3;
```

Tekkinud grupid saab mugavasti läbi käia foreach-tsükliga. Lihtsamal juhul ei pea me muutujatüüpide peale mõtlema ning võime tüübiks määrata var - selle peale otsib kompilaator ise sobiva tüübi andmetega ringi käimiseks. Grupi puhul on üheks väljaks Key, mis näitab tunnuse väärtust, mille järgi grupeeriti. Grupp ise on IEnumerable tüüpi andmestik, millest näiteks Joini abil võin väärtused kergesti tekstina välja trükkida. Selle poole võimalik samuti foreach-tsükli abil pöörduda või siis ToArray käskluse abil tulemus massiivi välja võtta nagu hiljem tehtud. Siin aga trükitakse lihtsalt välja, millised väärtused iga võtme alla jäid.

```
foreach(var grupp in grupid){
  Console.WriteLine("Jääk "+grupp.Key+" arvud "+String.Join(" ",grupp));
}
```

Toimuva üle rohkem kontrolli hoidmiseks on võimalik andmetüübid ka selgemalt määrata. Tüüp IGrouping sisaldab endas andmete võtmeid ja väärtusi. Praegusel juhul IGrouping<int, int> tähendab, et võti (ehk jääk jagamisel kolmega) on tüübist int ning võtmele vastava väärtustejada elemendid samuti tüübist int. Kui näiteks võtmeks oleks pikkus sentimeetrites ning väärtustejadaks selle pikkusega inimesed, siis oleks tüübiks IGrouping<int, string>. GroupBy annab tulemusena kõigepealt selliste IGrouping-tüüpi võtmele vastavate väärtusteahelate IEnumerable-tüüpi loetelu.

```
IEnumerable<IGrouping<int, int>> grupid2=arvud.GroupBy(arv => arv % 3);
```

Edasi on loetelust võimalik üksikud elemendid tsükli (või näiteks Where-käsu) abil välja võtta ning nende andmeid pruukima hakata. Juures ka näide, kuidas võtmele vastavad väärtused massiivina kätte saada ning neist esimene võtta.

```
foreach(IGrouping<int, int> grupp in grupid2){
  Console.WriteLine("Jääk "+grupp.Key+"  arvud "+String.Join(" ",grupp));
  int[] grupiarvud=grupp.ToArray<int>();
  Console.WriteLine("Esimene: "+grupiarvud[0]);
}
```

Näite kood tervikuna:

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
public class Grupid{
  public static void Main(string[] arg){
     IEnumerable<int> arvud=Enumerable.Range(101, 20); //alates 101st 20 arvu
       Console.WriteLine(String.Join(" ", arvud));
       var grupid=from arv in arvud group arv by arv%3;
       foreach(var grupp in grupid){
         Console.WriteLine("Jääk "+grupp.Key+" arvud "+String.Join(" ",grupp));
       }

       IEnumerable<IGrouping<int, int>> grupid2=arvud.GroupBy(arv => arv % 3);
       foreach(IGrouping<int, int> grupp in grupid2){
         Console.WriteLine("Jääk "+grupp.Key+"  arvud "+String.Join(" ",grupp));
         int[] grupiarvud=grupp.ToArray<int>();
         Console.WriteLine("Esimene: "+grupiarvud[0]);
       }
  }
}
/*
E:\jaagup\11\09\linq>Grupid
101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
Jääk 2 arvud 101 104 107 110 113 116 119
Jääk 0 arvud 102 105 108 111 114 117 120
Jääk 1 arvud 103 106 109 112 115 118
Jääk 2  arvud 101 104 107 110 113 116 119
Esimene: 101
Jääk 0  arvud 102 105 108 111 114 117 120
Esimene: 102
Jääk 1  arvud 103 106 109 112 115 118
Esimene: 103
*/
```

Ülesandeid

* Koosta eesnimede loetelu
* Grupeeri nimed pikkuse järgi
* Näita tulemused välja (pikkus, nimed ...)

* Näita, mitu nime iga pikkusega on
* Näita igast paarisarvulise nimepikkusega grupist välja esimene nimi.

Anonüümne tüüp

Soovides andmed ühe programmiosa juures kokku grupeerida, pole selleks vaja alati eraldi klassi või struktuuri looma hakata . Käsklus new lubab looksulgude vahel andmetüübi lennult tekitada ning muutujatüüp var suudab selle kinni püüda. Näites tehakse objekt p väljadega x ja y. Ning väljatrükil saab x-i väärtuse täiesti kätte.

```
using System;
public class AnonyymneTyyp{
  public static void Main(string[] arg){
    var p=new{x=3, y=5};
    Console.WriteLine(p.x);
  }
}
/*
E:\jaagup\11\09\linq>AnonyymneTyyp.exe
3
*/
```

Sellise tüübi saab mugavalt ära kasutada näiteks tekstifailist andmeid lugedes. Põhjalikumal ja viisakamal juhul tasuks lapse tarvis teha omaette klass ning temast eksemplar luua ning sinna sisse andmeid panna ja küsida. Kui aga ainult korraks vaja andmeid mällu lugeda ja töödelda, siis saab ka lihtsamalt.
Järgnevas näites võtab klassi File käsklus ReadAllLines failist nimepikkused.txt välja kõik seealsed read. Fail ise järmine

```
nimepikkused.txt
Juku 173
Mari 165
Kati 170
Madis 173
Mati 173
Aadu 175
Anu 165
```

Kuna massiiv vastab IEnumerable reeglitele, siis saab selle panna ka LINQ-päringu sisse. Et andmeid oleks kergem hiljem töödelda jagame rea tühiku kohalt kaheks ning moodustame uue objekti. Rea esimesest poolest saab väli nimega eesnimi, teine pool muudetakse int.Parse-käsklusega täisarvuks ning läheb vastava tüübiga väljaks pikkus. Substring aitab teksti tükeldada, IndexOf(" ") teatab tühiku asukoha. Automaatselt tekkis tüüp millel väljad eesnimi ja pikkus ning neid on võimalik edaspidiste arvutuste juures kasutada - praegu siis väljatrükiks.

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
using System.IO;
public class Grupifail{
  public static void Main(string[] argumendid){
     var lapsed=from rida in File.ReadAllLines("nimedpikkused.txt")
         select new{
              eesnimi=rida.Substring(0, rida.IndexOf(" ")),
              pikkus=int.Parse(rida.Substring(rida.IndexOf(" ")))
         };
     foreach(var laps in lapsed){
```

```
            Console.WriteLine(laps.eesnimi+" pikkusega "+laps.pikkus);
        }
    }
}
/*
E:\jaagup\11\09\linq>Grupifail
Juku pikkusega 173
Mari pikkusega 165
Kati pikkusega 170
Madis pikkusega 173
Mati pikkusega 173
Aadu pikkusega 175
Anu pikkusega 165
*/
```

Sama andmestiku saab mõnevõrra mugavamalt sisse lugeda avaldiste keelt kasutades. File.ReadAllLines sobib taas andmemassiivi allikaks. Reast kahe väärtuse kätte saamiseks on mugav see kõigepealt tühiku koha pealt massiiviks tükeldada ning siis edasi kummastki massiivielemendist objektiväli teha. Ülal näites tulnuks selle tarbeks uus lause kirjutada, siin aga saab avaldisi ritta sättides öelda, et kõigepealt tuleb ühes Select-is rida Split-i abil tühiku koha pealt tükeldada. Tulemusena on igale reale vastavas elemendis üks kaheelemendiline massiiv. Järgmise Selecti abil saab luua uue objekti määrates massiivi 0-indeksiga liikme eesnimeks ning järgmise pärast täisarvuks muutmist perekonnanimeks.

```
var lapsed=File.ReadAllLines("nimedpikkused.txt").Select(rida => rida.Split(' ')).
       Select(m => new{eesnimi=m[0], pikkus=int.Parse(m[1])});
```

Edasi võib objektikogumiga käituda nagu mujalgi, näiteks järjestades nad pikkuse järjekorda.

```
var lapsedPikkusteJarjekorras=lapsed.OrderBy(laps => laps.pikkus);
```

Tsükli abil saab need sealt ka ilusti kätte.

```
foreach(var laps in lapsedPikkusteJarjekorras){
    Console.WriteLine(laps.eesnimi+" pikkusega "+laps.pikkus);
}
```

Ka faili kirjutamine läheb üllatavalt valutult. File.WriteAllLines soovib andmete kirjutamiseks stringide kogumit. Select-käsu abil vormistame iga lapse objektist teksti mille algul on lapse pikkus, järgnevad tühik ja eesnimi.

```
File.WriteAllLines("sortpikkused.txt",
    lapsedPikkusteJarjekorras.Select(laps => laps.pikkus+" "+laps.eesnimi));
```

Ka grupeerimise kannatab suuremate muredeta ette võtta. Laste andmete kümnendite kaupa grupeerimiseks saab võtmeks täisosa pikkuse suhtest kümnega (st. 167 puhul nt. 16). Mis omakorda kümnega korrutatud, et pärisandmeid mugavam lugeda oleks.

Edasi saab gruppidega toimetada juba sarnaselt kui eelmises näites. Laste nimede stringimassiivina väljavõtmise juures aitab samuti Select eesnimede eristamiseks ning ToArray massiiviks muutmiseks (kui seda peaks parajasti vaja minema). Tsüklimuutuja kymnend vastab ise IEnumerable reeglitele ning Count() annab näiteks elementide arvu. Laste andmete sobival kujul tekstiks vormistamist toetab taas Select.

```
var lapsedPikkusKymnenditena=lapsed.GroupBy(laps => laps.pikkus/10*10);
foreach(var kymnend in lapsedPikkusKymnenditena){
   string[] eesnimed=kymnend.Select(laps => laps.eesnimi).ToArray<string>();
   Console.WriteLine(kymnend.Key+", "+kymnend.Count()+" last: "+
```

```
                String.Join(", ", kymnend.Select(laps => laps.eesnimi+"-"+laps.pikkus)));
        }
```

Näide tervikuna:

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
using System.IO;
public class Grupifail2{
  public static void Main(string[] argumendid){
     var lapsed=File.ReadAllLines("nimedpikkused.txt").Select(rida => rida.Split(' ')).
            Select(m => new{eesnimi=m[0], pikkus=int.Parse(m[1])});
     var lapsedPikkusteJarjekorras=lapsed.OrderBy(laps => laps.pikkus);
        foreach(var laps in lapsedPikkusteJarjekorras){
            Console.WriteLine(laps.eesnimi+" pikkusega "+laps.pikkus);
        }
        File.WriteAllLines("sortpikkused.txt",
            lapsedPikkusteJarjekorras.Select(laps => laps.pikkus+" "+laps.eesnimi));
        var lapsedPikkusKymnenditena=lapsed.GroupBy(laps => laps.pikkus/10*10);
        foreach(var kymnend in lapsedPikkusKymnenditena){
           string[] eesnimed=kymnend.Select(laps => laps.eesnimi).ToArray<string>();
           Console.WriteLine(kymnend.Key+", "+kymnend.Count()+" last: "+
                String.Join(", ", kymnend.Select(laps => laps.eesnimi+"-"+laps.pikkus)));
        }

  }
}
/*
E:\jaagup\11\09\linq>Grupifail2
Mari pikkusega 165
Anu pikkusega 165
Kati pikkusega 170
Juku pikkusega 173
Madis pikkusega 173
Mati pikkusega 173
Aadu pikkusega 175
170, 5 last: Juku-173, Kati-170, Madis-173, Mati-173, Aadu-175
160, 2 last: Mari-165, Anu-165

E:\jaagup\11\09\linq>type nimedpikkused.txt
Juku 173
Mari 165
Kati 170
Madis 173
Mati 173
Aadu 175
Anu 165

E:\jaagup\11\09\linq>type sortpikkused.txt
165 Mari
165 Anu
170 Kati
173 Juku
173 Madis
173 Mati
175 Aadu
*/
```

Ülesandeid

* Koosta tekstifail, kus ühes tulbas kirjas inimese eesnimi ning teises linn, kus ta sündinud
* Loe andmed objektidena mällu
* Väljasta inimeste eesnimed

* Sorteeri andmed linnanimede järjekorras, väljasta teise faili
* Väljasta erinevad linnade nimed
* Väljasta iga linna kohta, mitu inimest selles sündinud on
* Väljasta ka igas linnas sündinud inimeste nimed

Laiendusmeetodid

Koos LINQ tulekuga ilmus C#-keelde tore täiendus - käsklusi on võimalik lisada ka klassidele, millest me otse pärida ei saa või ei taha. Nõndamoodi on massiividele ja muudele IEnumerable-liidesele vastavatele tüüpidele lisatud hulk kasulikke käsklusi. Sarnaselt saab aga oma rakenduse kontekstis muudki tüüpi objektide oskusi täiendada.

Meetodeid saab juurde lisada staatilise klassi abil, mille staatilise meetodi esimesele parameetrile on ette lisatud võtmesõna this. Kui vastav nimeruum hiljem using-käsuga külge võetakse, siis selle usingu skoobis käitub lisatud meetod nagu oleks ta vastava klassi isendimeetod. Ehk siis näitena siin luuakse nimeruumis LaiendusMeetodid staatilise klassi sisse käsklus Pikkus, mis saab parameetriks stringi.

fail: LaiendusMeetodid.cs
```
using System;
namespace LaiendusMeetodid{
  public static class TekstiMeetodid{
     public static int Pikkus(this String s){
           return s.Length;
        }
  }
}
```

Kui järgnevas failis eelnev nimeruum külge võetakse, siis on kõikidel stringidel automaatselt küljes käsklus nimega Pikkus().

fail: LaiendusProov.cs
```
using System;
using LaiendusMeetodid;
public class LaiendusProov{
  public static void Main(string[] arg){
    Console.WriteLine("Tere".Pikkus());
  }
}
/*
E:\jaagup\11\09\linq>LaiendusProov
4
*/
```

Omapärane, et ei loe, millise nimega klassi sisse vastav laiendusmeetod pandud on. Kiusatusena tekkis mõte kontrollida, mis siis saab, kui samanimeline laiendusmeetod on defineeritud mitme sama nimeruumi klassi sees. Tulemuseks oli kompilaatoripoolne veateade, mis andis märku, et enam pole selge, millist käsklust käivitada tahetakse.

Juurde näitena mõned laiendusmeetodid, millest ehk rohkem abi võiks olla. ArvudeSumma paneb tekstile külge käskluse, mis jagab teksti sõnadeks ning liidab kokku need, mis arvud on.

Ka tavalisele int-tüübile saab käsklusi juurde lisada. Siin näites astendamiskäskluse.

Kolmanda funktsioonina on toodud näide massiivile oskuste lisamiseks. Käsklus tühjenda muudab

etteantud täisarvulise massiivi kõik väärtused nullideks.

fail: LaiendusMeetodid2.cs

```
using System;
namespace LaiendusMeetodid{
   static class TekstiMeetodid2{
      public static int ArvudeSumma(this String s){
            string[] m=s.Split(' ');
               int summa=0;
               int abi=0;
               foreach(string sona in m){
                 if(int.TryParse(sona, out abi)){
                   summa+=abi;
                 }
               }
               return summa;
         }
   }
   static class ArvuMeetodid{
     public static int astmes(this int arv, int aste){
           int abi=1;
              for(int i=1; i<=aste; i++){
                 abi*=arv;
              }
              return abi;
        }
        public static void Tyhjenda(this int[] m){
           for(int i=0; i<m.Length; i++){
                m[i]=0;
              }
        }
   }
}
```

Käivitamisel võib laiendusmeetodeid külge võtta mitmest failist, peaasi, et nimeruum klapib. Siingi näites on Pikkus()-nimeline meetod pärit failist LaiendusMeetodid.cs, muud lisandused aga failist LaiendusMeetodid2.cs. Nii tuleb hoolitseda, et kompilaatorile kõik vajalikud tükid kättesaadavad oleksid - olgu siis käsureal failinimesid ette andes või arenduskeskkonnas projekti juurde kuuluvaid faile määrates.

fail: LaiendusProov2.cs

```
using System;
using LaiendusMeetodid;
public class LaiendusProov2{
  public static void Main(string[] arg){
    Console.WriteLine("Tere".Pikkus());
       Console.WriteLine("5 pirni ja 12 õuna".ArvudeSumma());
       Console.WriteLine(3.astmes(2));
       int[] arvud={2,4, 5};
       arvud.Tyhjenda();
       Console.WriteLine(String.Join(" ", arvud));
  }
}
/*
E:\jaagup\11\09\linq>csc LaiendusMeetodid.cs LaiendusMeetodid2.cs LaiendusProov2.cs
Microsoft (R) Visual C# 2010 Compiler version 4.0.30319.1
Copyright (C) Microsoft Corporation. All rights reserved.

E:\jaagup\11\09\linq>LaiendusProov2
4
17
9
0 0 0
*/
```

Ülesandeid

* Loo stringile laiendusmeetod esitähe väljastamiseks
* Loo stringile laiendusmeetod, mis väljastaks seal sees olnud arvud massiivina
* Loe arvumassiivile laiendusmeetod, mille abil on võimalik seal sees olnud arve nihutada etteantud sammu jagu
* Loo stringile laiendusmeetodid samanimelise faili sisu lugemiseks või kirjutamiseks.
* Koosta/leia omaloodud klass. Lisa sellele oskusi laiendusmeetodi abil.

Üldisi ülesandeid

* Tutvu nimeruumi System.IO klasside Directory ning File oskustega failinimede ja sisude kuvamisel
* Näita välja kõik kataloogis olevad failinimed
* Näita välja nimed koos pikkustega
* Sorteeri failinimed pikkuste järjekorras
* Näita kõigi tekstifailide pikkused
* Näita tekstifailide pikkuste summa
* Näita iga tekstifaili kohta seal olevate ridade arv
* Näita iga tekstifaili kohta seal olevate sõnade arv
* Liida kokku kõik kataloogi tekstifailides olevate arvude väärtused
* Koosta rekursiivne alamprogramm leidmaks kõigi tekstifailides olevate arvude summa etteantud kataloogis ning selle alamkataloogides. Katseta tulemust.

Viiteid

Heiki Tähise loodud õpiobjekt
http://www.e-ope.ee/_download/euni_repository/file/669/LINQ.zip/LINQ/index.html

Eestikeelsest õppematerjalist leiab väärt selgitusi LINQ ja XMLi koos kasutamise kohta. Samuti tutvutakse paindliku var-muutujatüübi ning käigu pealt loodavate anonüümsete objektidega.

Microsofti ametlik tutvustusleht
http://msdn.microsoft.com/en-us/library/bb308959.aspx

Ametlikul lehel on heade selgitavate näidetega läbi käidud pea kõik LINQ põhilised käsud, mis andmekogumina otse rakendada saab, samuti tehtud nendest hästi loetav kokkuvõte.