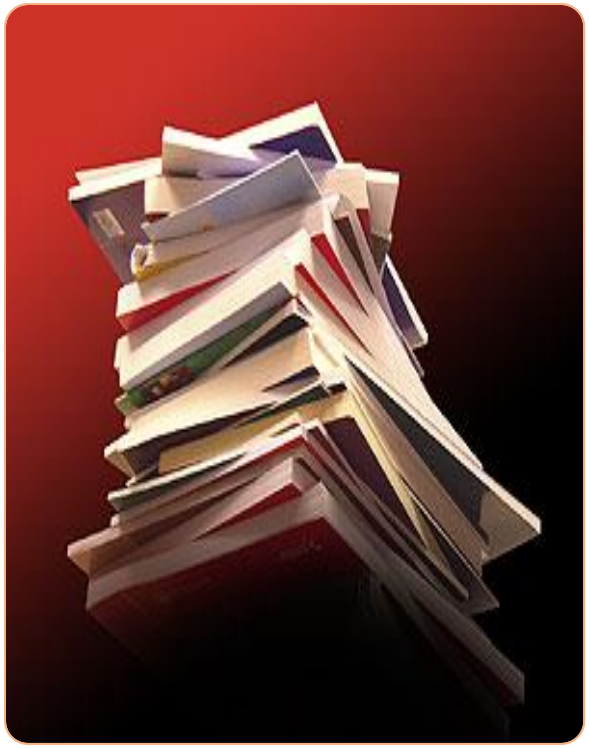
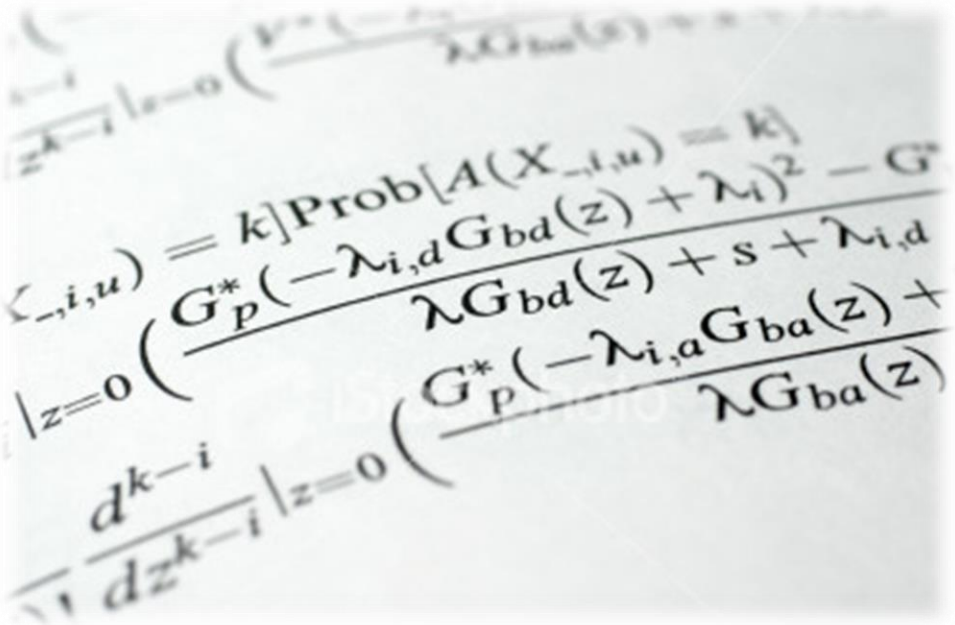
Lambda Expressies en LINQ

Inhoud

1. Lambda Expressies
2. LINQ Queries



Lambda Expressies



Lambda Expressies

- Een lambda expressie is een anonymous methode die expressies en statements bevat
  - Wordt gebruikt voor delegates of expression- tree types
- Lambda expressies
  - Gebruik van lambda operator =>
    - Lees als "gaat naar"
  - Aan de linker zijde van deze operator worden de invoer parameters gespecifieerd
  - Aan de rechter zijde van deze operator worden expressies of statements gespecifieerd

Lambda Expressions – Voorbeelden

- Wordt meestal gebruik bij collection extension methods, zoals `FindAll()` en `RemoveAll()`

`Voorbeelden:`

```
List<int> list = new List<int>() { 1, 2, 3, 4 };
List<int> evenNumbers = list.FindAll(x => (x % 2) == 0);
foreach (var num in evenNumbers)
{
       Console.Write("{0} ", num);
}
Console.WriteLine();  // 2 4

list.RemoveAll(x => x > 3); // 1 2 3

list.ForEach(n => Console.Write(n + "-"));// 1 2 3
```

Sorteren met Lambda Expressie

```
var honden = new Hond[]
{
                    new Hond { Naam="Sharo", Leeftijd=8 },
                    new Hond { Naam="Rex", Leeftijd=4 },
                    new Hond { Naam="Strela", Leeftijd=1 },
                    new Hond { Naam="Bora", Leeftijd=3 }
};
var gesorteerdeLijst = honden.OrderBy(pet => pet.Leeftijd);
foreach (Hond hond in gesorteerdeLijst)
{
        Console.WriteLine("{0} -> {1}",
                hond.Naam, hond.Leeftijd);
}
```

Lambda Code Expressies

```
List<int> list = new List<int>() { 20, 1, 4, 8, 9, 44 };

List<int> evenGetallen = list.FindAll((i) =>
{
   Console.WriteLine("waarde van i is: {0}", i); //elk element aflopen
   return (i % 2) == 0;  //geef enkel even getallen terug
});

Console.WriteLine("Hier zijn de even getallen:");
foreach (int even in evenGetallen)
   Console.Write("{0}\t", even);
```

Delegates die Lambda Functies bij-houden

- Lambda functions kunnen bijgehouden worden in variablen van type delegate
  - Delegates zijn 'getypeerde referenties' naar functies
- Standaard function delegate in .NET (generic):
  - `Func<TResult>`, `Func<T, TResult>`, `Func<T1, T2, TResult>`, …
- Voorbeelden:

```
Func<bool> boolFunc = () => true;
Func<int, bool> intFunc = (x) => x < 10;
if (boolFunc() && intFunc(5))
    Console.WriteLine("5 < 10");
```

Predicates

- Predicates zijn voorgedefinieerde delegates met de volgende signatuur (generieke delegate die bool teruggeeft):

```
public delegate bool Predicate<T>(T obj)
```

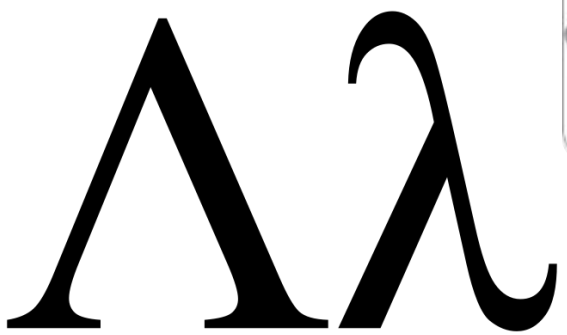
  - Definieert een manier om te controleren of een object voordoet aan een Boolean criterium
  - Analoog aan `Func<T, bool>`
- Wordt door meerdere extension methods van `Array` and `List<T>` gebruikt om te zoeken naar een element
  - Voorbeeld :`List<T>.FindAll(…)` zoekt alle elementen die aan bepaalde criteria voldoet

Predicates – Voorbeeld

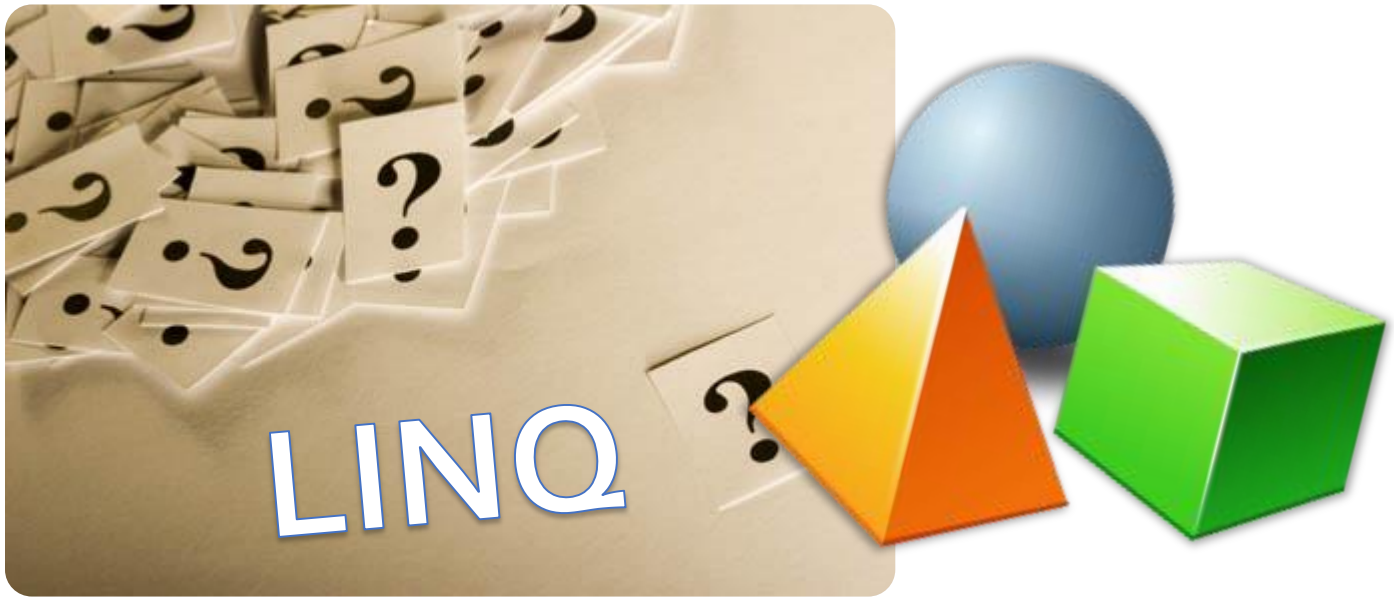
```
List<string> gemeenten = new List<string>()
{
        "Gent", "Antwerpen", "Brussel", "Kortijk", "Roeselare"
};

List<string> gemeentenMetS =
        gemeenten.FindAll(delegate (string gemeente)
         {
                  return gemeente.Contains("s");
        });

// Zelfde maar korter met lambda expression
List<string> ookGemeentenMetS =
          gemeenten.FindAll((gemeente) => gemeente.Contains("s"));

foreach (string gemeente in ookGemeentenMetS)
{
        Console.WriteLine(gemeente);
}
```

Lambda Expressies

- Live Demo



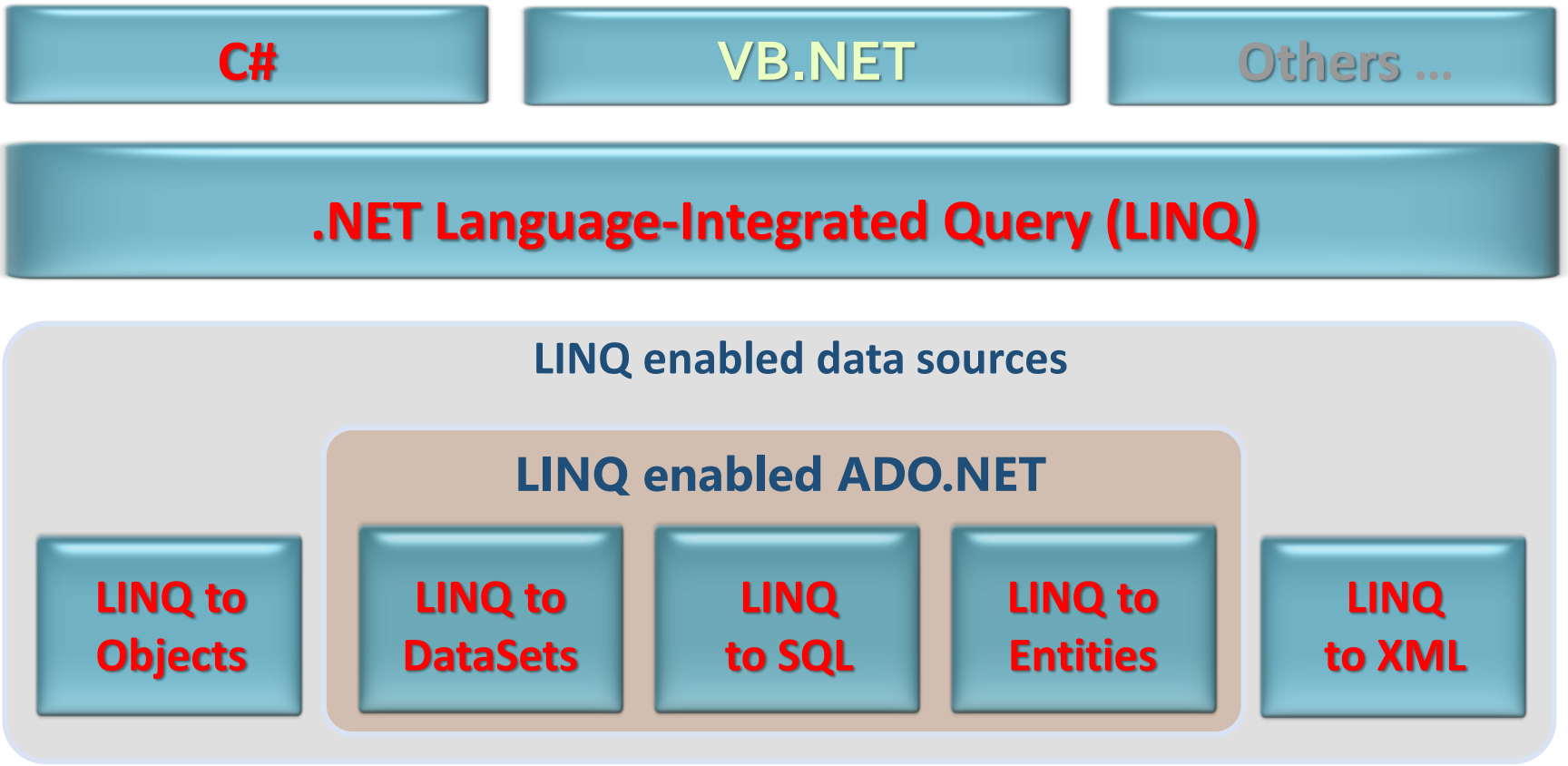
LINQ en Query Keywords



Bouwstenen van LINQ

- LINQ is een een verzameling extensies van .NET
- Omvat language-geïntegreerde query, en transformatieve operaties
  - Heeft consistente manier om "data" op te vragen en te wijzigen
- **Query expressies** kunnen direct in C# code gedefinieerd worden
- Wordt gebruikt voor interactie met meerdere verschillende data types
  - Data types worden geconverteerd naar expression trees op compile time en geëvalueerd tijdens runtime

LINQ to *


C#
VB.NET
Others ...
.NET Language-Integrated Query (LINQ)
LINQ enabled data sources
LINQ enabled ADO.NET
LINQ to Objects
LINQ to DataSets
LINQ to SQL
LINQ to Entities
LINQ to XML
<book>
<title/>
<author/>
<price/>
</book>
Objects
Relational Data
XML

LINQ en Query Keywords

- Language Integrated Query (LINQ) query keywords
  - **`from`** – specifieert data source of range
  - **`where`** – filtert source elementen
  - **`select`** – specifieert het type en vorm van de elementen in de teruggegeven sequentie
  - **`groupby`** – groepeert query results volgens een gegeven key-value
  - **`orderby`** – sorteert query resultaten in alfabetische volgorde/klein naar groot (ascending) of

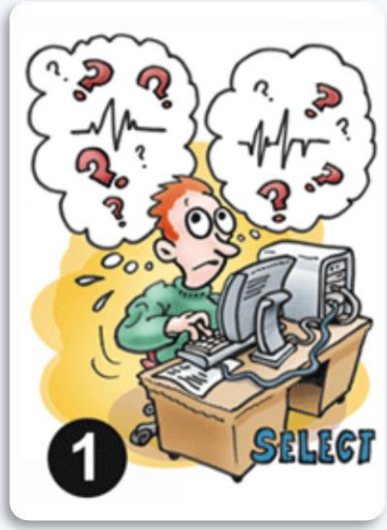
    in descending (groot naar klein)

Query Keywords – Voorbeelden

```
int[] getallen = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
```

- **select, from and where clauses:**

```
var queryKleineGetallen =

        from getal in getallen
        where getal < 5
        select getal;

foreach (var getal in queryKleineGetallen)
{
        Console.Write(getal.ToString() + " ");
}
            // 4 1 3 2 0
```



Query Keywords – Voorbeelden(2)

- **Geneste queries:**

```
 string[] gemeenten = { "Brussel", "Antwerpen", "Brussel", "Gent",
"Kortrijk" };

var paren =
                from t1 in gemeenten
                from t2 in gemeenten
                select new { T1 = t1, T2 = t2 };

foreach (var gemeentePaar in paren)
{
        Console.WriteLine("({0}, {1})",
        gemeentePaar.T1, gemeentePaar.T2);
}
```

Query Keywords – Voorbeeld(3)

```
string[] vruchten =  { "kers", "appel", "bosbes", "banaan" };
```

- **Sorteren met orderby:**

```
// Sorteren in alfabetische volgorde
var fruitAlfabetisch =
        from fruit in vruchten
        orderby fruit
        select fruit;

foreach (string fruit in fruitAlfabetisch)
{
         Console.WriteLine(fruit);
}
```

Standaard Query Operators - Voorbeeld

```
 string[] spelletjes = {"Roborally", "BioShock","Half Life",
  "The Darkness","Daxter", "System Shock 2"};

// Linq query expressie in Method syntax
//die gebruik maakt van extension methoden
// van Array via Enumerable type
var subset = spelletjes.Where(spel => spel.Length > 6).
       OrderBy(spel => spel).Select(spel => spel);

foreach (var spel in subset)
       Console.WriteLine(spel);
Console.WriteLine();
```

```
var subset =
  from s in spelletjes
  where s.Length > 6
  orderby s
  select s;
```

Voorbeeld: aantal woorden in een string teruggeven via LINQ

```
string text = "Trump News en fake news ...";

string zoekTerm = "news";
string[] woorden = text.Split(
  new char[] { '.', '?', '!', ' ', ';', ':', ',' },
    StringSplitOptions.RemoveEmptyEntries);

// Gebruik ToLower() om  zowel "news" en bv "News" terug te krijgen
var zoekQuery =
              from woord in woorden
              where woord.ToLower() == zoekTerm.ToLower()
              select woord;
int aantalWoorden = zoekQuery.Count();
Console.WriteLine("Aantal keer dat zoekTerm voorkomt:" +
aantalWoorden);
```

```
int aantalWoorden = woorden.Where(
  w => w.ToLower() ==
  zoekTerm.ToLower()).Count();
```

Linq queries op Arrays

- Alle soorten arrays kunnen worden gebruikt bij LINQ
  - Kan ook een niet-getypeerde array van objecten zijn
  - Queries kunnen ook op custom objects worden toegepast

Voorbeeld:

```
var titels =
  from b in boeken
  where b.Titel.Contains("Action")
  select b.Titel;
```

```
Boek[] boeken = {
  new Boek { Titel= "LINQ in Action" },
  new Boek { Titel= "LINQ for Fun" },
  new Boek { Titel= "Extreme LINQ" } };
var titels = boeken
              .Where(book => book.Titel.Contains("Action"))
              .Select(book => book.Titel);
List<string> boekTitels = titels.ToList();
boekTitels.ForEach(s => Console.WriteLine(s));
```

Linq queries op Generieke Lists

- Het voorgaande voorbeeld kan worden aangepast zodat deze werkt met een generic list, zoals
  - `List<T>`, `LinkedList<T>`, `Queue<T>`, `Stack<T>`, `HashSet<T>`, …

```
 List<Boek> boeken = new List<Boek>() {
                new Boek { Titel = "LINQ in Action" },
                new Boek { Titel = "LINQ for Fun" },
                new Boek { Titel = "Extreme LINQ" } };
var titels = boeken
                .Where(b => b.Titel.Contains("Action"))
                .Select(b => b.Titel);
```

Linq queries op Strings

- `System.String` is ook een sort collectie
  - Omdat deze class `IEnumerable<char>` implementeert
- String objecten kunnen dus ook gequeried worden met LINQ to Objects, zoals alle collecties

```
string tekst = "Aantal niet-letters in deze tekst: 8";
var aantal = tekst
                  .Where(c => !Char.IsLetter(c))
                  .Count();
Console.WriteLine(aantal);//  8
```

```
var count =
  (from c in tekst
  where !Char.IsLetter(c)
  select c).Count();
```

Methoden voor aggregatie

- `Average()`
  - Berekent de gemiddelde waarde van de elementen uit een collectie
- `Count()`
  - Telt het aantal elementen in een collectie
- `Max()/Min()`
  - Geeft de maximum/minimum waarde in een collectie
- `Sum()`
  - Sommeert de waarden van de elementen in een collectie

Aggregation Methoden – Voorbeelden

- Count(<conditie>)

```
double[] temperaturen =  {28.0, 19.5, 32.3, 33.6, 26.5, 29.7};
int aantalHogeTemperaturen = temperaturen.Count(p => p > 30);
Console.WriteLine(aantalHogeTemperaturen); // 2
```

```
var hoogsteTempCount =
  (from t in temperaturen
   where t > 30
   select t).Count();
```

- Max()

```
double[] temperaturen =  {28.0, 19.5, 32.3, 33.6, 26.5, 29.7};
double maxTemp = temperaturen.Max();
Console.WriteLine(maxTemp);//  33.6
```

```
var maxTemp =
  (from t in temperaturen
   select t).Max();
```

LINQ Query Keywords

- Demo

Lambda Expressions en LINQ

Vragen?

Oefeningen

1. Schrijf een methode die voor een gegeven array van studenten alle studenten teruggeeft wiens voornaam alfabetisch vóór zijn familienaam komt/ Gebruik LINQ query operatoren.
2. Schrijf een LINQ query die de voornaam en familienaam teruggeeft van alle studenten tussen 18 en 24 jaar.

Referenties

- C# Programming @ Telerik Academy
  - csharpfundamentals.telerik.com
- Telerik Software Academy
  - academy.telerik.com