

# Curso C# Completo

## Programação Orientada a Objetos + Projetos

**Capítulo: Expressões lambda, delegates e LINQ**

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Uma experiência com Comparison<T>

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Problema

- Suponha uma classe Product com os atributos name e price.  
Suponha que precisamos ordenar uma lista de objetos Product.
- Podemos implementar a comparação de produtos por meio da implementação da interface IComparable<Product>
- Entretanto, desta forma nossa classe Product não fica fechada para alteração: se o critério de comparação mudar, precisaremos alterar a classe Product.
- Podemos então usar outra sobrecarga do método "Sort" da classe List:  
`public void Sort(Comparison<T> comparison)`

Product
- name : String
- price : Double

## Comparison<T> (System)

[https://msdn.microsoft.com/en-us/library/tfakywbh\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/tfakywbh(v=vs.110).aspx)

```
public delegate int Comparison<in T>(T x, T y);
```

Método Sort com Comparison<T> da classe List:

<https://msdn.microsoft.com/en-us/library/w56d4y5z%28v=vs.110%29.aspx>

## Resumo da aula

```
public void Sort(Comparison<T> comparison)
```

- Referência simples de método como parâmetro
- Referência de método atribuído a uma variável tipo delegate
- Expressão lambda atribuída a uma variável tipo delegate
- Expressão lambda inline

<https://github.com/acenelio/lambda1-csharp>

## Programação funcional e cálculo lambda

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Paradigmas de programação

- **Imperativo** (C, Pascal, Fortran, Cobol)
- **Orientado a objetos** (C++, Object Pascal, Java (< 8), C# (< 3))
- **Funcional** (Haskell, Closure, Clean, Erlang)
- **Lógico** (Prolog)
- **Multiparadigma** (JavaScript, Java (8+), C# (3+), Ruby, Python, Go)

## Paradigma funcional de programação

Baseado no formalismo matemático Cálculo Lambda (Church 1930)

	<b>Programação Imperativa</b>	<b>Programação Funcional</b>
Como se descreve algo a ser computado (*)	comandos ("como" - imperativa)	expressões ("o quê" - declarativa)
Funções possuem transparência referencial (ausência de efeitos colaterais)	fraco	forte
Objetos imutáveis (*)	raro	comum
Funções são objetos de primeira ordem	não	sim
Expressividade / código conciso	baixa	alta
Inferência de tipos	raro	comum
Execução tardia (lazy)	raro	comum

# Transparência referencial

Uma função possui transparência referencial se seu resultado for sempre o mesmo para os mesmos dados de entrada. Benefícios: simplicidade e previsibilidade.

```
using System;

namespace Course {
    class Program {

        public static int globalValue = 3;

        static void Main(string[] args) {

            int[] vect = new int[] { 3, 4, 5 };
            ChangeOddValues(vect);
            Console.WriteLine(string.Join(" ", vect));
        }

        public static void ChangeOddValues(int[] numbers) {
            for (int i = 0; i < numbers.Length; i++) {
                if (numbers[i] % 2 != 0) {
                    numbers[i] += globalValue;
                }
            }
        }
    }
}
```



Exemplo de  
função que não é  
referencialmente  
transparente

## Funções são objetos de primeira ordem (ou primeira classe)

Isso significa que funções podem, por exemplo, serem passadas como parâmetros de métodos, bem como retornadas como resultado de métodos.

```
class Program {

    static int CompareProducts(Product p1, Product p2) {
        return p1.Name.ToUpper().CompareTo(p2.Name.ToUpper());
    }

    static void Main(string[] args) {

        List<Product> list = new List<Product>();

        list.Add(new Product("TV", 900.00));
        list.Add(new Product("Notebook", 1200.00));
        list.Add(new Product("Tablet", 450.00));

        list.Sort(CompareProducts);

        (...)
    }
}
```

## Inferência de tipos

```
List<Product> list = new List<Product>();

list.Add(new Product("TV", 900.00));
list.Add(new Product("Notebook", 1200.00));
list.Add(new Product("Tablet", 450.00));

list.Sort((p1, p2) => p1.Name.ToUpper().CompareTo(p2.Name.ToUpper()));

foreach (Product p in list) {
    Console.WriteLine(p);
}
```

## Expressividade / "como" vs. "o quê"

```
int sum = 0;
foreach (int x in list) {
    sum += x;
}
```

**VS.**

```
int sum = list.Aggregate(0, (x, y) => x + y);
```

# O que são "expressões lambda"?

Em programação funcional, expressão lambda corresponde a uma função anônima de primeira classe.

```
class Program {  
    static int CompareProducts(Product p1, Product p2) {  
        return p1.Name.ToUpper().CompareTo(p2.Name.ToUpper());  
    }  
  
    static void Main(string[] args) {  
        (...)  
        list.Sort(CompareProducts);  
        list.Sort((p1, p2) => p1.Name.ToUpper().CompareTo(p2.Name.ToUpper()));  
        (...)  
    }  
}
```

## Resumo da aula

	Programação Imperativa	Programação Funcional
Como se descreve algo a ser computado (*)	comandos ("como" - imperativa)	expressões ("o quê" - declarativa)
Funções possuem transparência referencial (ausência de efeitos colaterais)	fraco	forte
Objetos imutáveis (*)	raro	comum
Funções são objetos de primeira ordem	não	sim
Expressividade / código conciso	baixa	alta
Tipagem dinâmica / inferência de tipos	raro	comum
Execução tardia (lazy)	raro	comum

Cálculo Lambda = formalismo matemático base da programação funcional

Expressão lambda = função anônima de primeira classe

# Introdução a delegates

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Delegates

- <https://docs.microsoft.com/en-us/dotnet/standard/delegates-lambdas>
- É uma referência (com type safety) para um ou mais métodos
  - É um tipo referência
- Usos comuns:
  - Comunicação entre objetos de forma flexível e extensível (eventos / callbacks)
  - Parametrização de operações por métodos (programação funcional)



## Delegates pré-definidos

- Action
- Func
- Predicate

## Demo

```
namespace Course.Services {  
    class CalculationService {  
        public static double Max(double x, double y) {  
            return (x > y) ? x : y;  
        }  
  
        public static double Sum(double x, double y) {  
            return x + y;  
        }  
  
        public static double Square(double x) {  
            return x * x;  
        }  
    }  
}
```

## Demo

```
using System;
using Course.Services;

namespace Course {

    delegate double BinaryNumericOperation(double n1, double n2);

    class Program {
        static void Main(string[] args) {
            double a = 10;
            double b = 12;

            // BinaryNumericOperation op = CalculationService.Sum;
            BinaryNumericOperation op = new BinaryNumericOperation(CalculationService.Sum);

            // double result = op(a, b);
            double result = op.Invoke(a, b);
            Console.WriteLine(result);
        }
    }
}
```

## Multicast delegates

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Multicast delegates

- Delegates que guardam a referência para mais de um método
- Para adicionar uma referência, pode-se usar o operador +=
- A chamada Invoke (ou sintaxe reduzida) executa todos os métodos na ordem em que foram adicionados
- Seu uso faz sentido para métodos void

## Demo

```
using System;

namespace Course.Services {
    class CalculationService {

        public static void ShowMax(double x, double y) {
            double max = (x > y) ? x : y;
            Console.WriteLine(max);
        }

        public static void ShowSum(double x, double y) {
            double sum = x + y;
            Console.WriteLine(sum);
        }
    }
}
```

```
using System;
using Course.Services;

namespace Course {

    delegate void BinaryNumericOperation(double n1, double n2);

    class Program {
        static void Main(string[] args) {
            double a = 10;
            double b = 12;

            BinaryNumericOperation op = CalculationService.ShowSum;
            op += CalculationService.ShowMax;
            op(a, b);
        }
    }
}
```

## Predicate (exemplo com RemoveAll)

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Predicate (System)

- Representa um método que recebe um objeto do tipo T e retorna um valor booleano

- <https://msdn.microsoft.com/en-us/library/bfcke1bz%28v=vs.110%29.aspx>

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

## Problema exemplo

Fazer um programa que, a partir de uma lista de produtos, remova da lista somente aqueles cujo preço mínimo seja 100.

```
List<Product> list = new List<Product>();  
  
list.Add(new Product("Tv", 900.00));  
list.Add(new Product("Mouse", 50.00));  
list.Add(new Product("Tablet", 350.50));  
list.Add(new Product("HD Case", 80.90));
```

<https://github.com/acenelio/lambda2-csharp>

# Action (exemplo com ForEach)

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Action (System)

- Representa um método void que recebe zero ou mais argumentos

- <https://msdn.microsoft.com/en-us/library/system.action%28v=vs.110%29.aspx>

```
public delegate void Action();  
public delegate void Action<in T>(T obj);  
public delegate void Action<in T1, in T2>(T1 arg1, T2 arg2);  
public delegate void Action<in T1, in T2, in T3>(T1 arg1, T2 arg2, T3 arg3);  
(16 sobrecargas)
```

## Problema exemplo

Fazer um programa que, a partir de uma lista de produtos, aumente o preço dos produtos em 10%.

```
List<Product> list = new List<Product>();  
  
list.Add(new Product("Tv", 900.00));  
list.Add(new Product("Mouse", 50.00));  
list.Add(new Product("Tablet", 350.50));  
list.Add(new Product("HD Case", 80.90));
```

<https://github.com/acenelio/lambda3-csharp>

## Func (exemplo com Select)

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Func (System)

- Representa um método que recebe zero ou mais argumentos, e retorna um valor

- <https://msdn.microsoft.com/en-us/library/bb534960%28v=vs.110%29.aspx>

```
public delegate TResult Func<out TResult>();  
public delegate TResult Func<in T, out TResult>(T obj);  
public delegate TResult Func<in T1, in T2, out TResult>(T1 arg1, T2 arg2);  
public delegate TResult Func<in T1, in T2, in T3, out TResult>(T1 arg1, T2 arg2, T3 arg3);
```

(16 sobrecargas)

## Problema exemplo

Fazer um programa que, a partir de uma lista de produtos, gere uma nova lista contendo os nomes dos produtos em caixa alta.

```
List<Product> list = new List<Product>();  
  
list.Add(new Product("Tv", 900.00));  
list.Add(new Product("Mouse", 50.00));  
list.Add(new Product("Tablet", 350.50));  
list.Add(new Product("HD Case", 80.90));
```

<https://github.com/acenelio/lambda4-csharp>



## Nota sobre a função Select

- A função "Select" (pertencente ao LINQ) é uma função que aplica uma função a todos elementos de uma coleção, gerando assim uma nova coleção (do tipo IEnumerable).

```
List<int> numbers = new List<int> { 2, 3, 4 };  
IEnumerable<int> newList = numbers.Select(x => 2 * x);  
Console.WriteLine(string.Join(" ", newList));
```

**4 6 8**

## Criando funções que recebem funções como argumento

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Recordando

- removeAll(Predicate)
- ForEach(Action)
- Select(Func)

## Problema exemplo

Fazer um programa que, a partir de uma lista de produtos, calcule a soma dos preços somente dos produtos cujo nome começa com "T".

```
List<Product> list = new List<Product>();  
  
list.Add(new Product("Tv", 900.00));  
list.Add(new Product("Mouse", 50.00));  
list.Add(new Product("Tablet", 350.50));  
list.Add(new Product("HD Case", 80.90));
```



1250.50

<https://github.com/acenelio/lambda5-csharp>

# Introdução ao LINQ

<http://educandoweb.com.br>

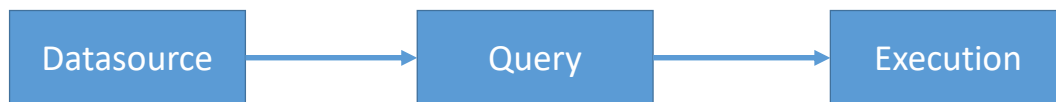
Prof. Dr. Nelio Alves

## LINQ - Language Integrated Query

- É um conjunto de tecnologias baseadas na integração de funcionalidades de consulta diretamente na linguagem C#
  - Operações chamadas diretamente a partir das coleções
  - Consultas são objetos de primeira classe
  - Suporte do compilador e IntelliSense da IDE
- Namespace: System.Linq
- Possui diversas operações de consulta, cujos parâmetros tipicamente são expressões lambda ou expressões de sintaxe similar à SQL
- Referência:
  - <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/index>

## Três passos

- Criar um data source (coleção, array, recurso de E/S, etc.)
- Definir a query
- Executar a query (foreach ou alguma operação terminal)



## Demo

```
// Specify the data source.  
int[] numbers = new int[] { 2, 3, 4, 5 };  
  
// Define the query expression.  
IEnumerable<int> result = numbers  
    .Where(x => x % 2 == 0)  
    .Select(x => 10 * x);  
  
// Execute the query.  
foreach (int x in result) {  
    Console.WriteLine(x);  
}
```

# Operações do LINQ / Referências

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Operações do LINQ

- **Filtering:** Where, OfType
- **Sorting:** OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse
- **Set:** Distinct, Except, Intersect, Union
- **Quantification:** All, Any, Contains
- **Projection:** Select, SelectMany
- **Partition:** Skip, Take
- **Join:** Join, GroupJoin
- **Grouping:** GroupBy
- **Generational:** Empty
- **Equality:** SequenceEquals
- **Element:** ElementAt, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault
- **Conversions:** AsEnumerable, AsQueryable
- **Concatenation:** Concat
- **Aggregation:** Aggregate, Average, Count, LongCount, Max, Min, Sum

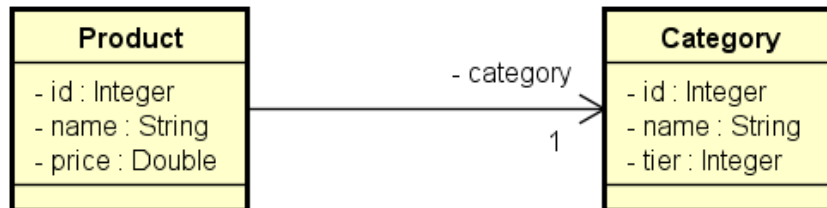
## Referências

- <https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b/view/SamplePack/1?sortBy=Popularity>
- <https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b/view/SamplePack/2?sortBy=Popularity>
- <https://odetocode.com/articles/739.aspx>

## Demo: LINQ com Lambda

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves



## Resumo da aula

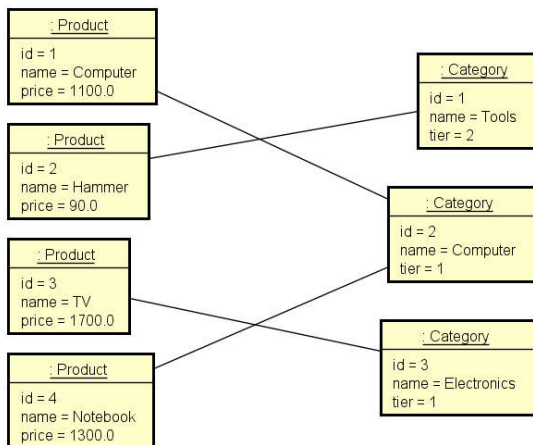
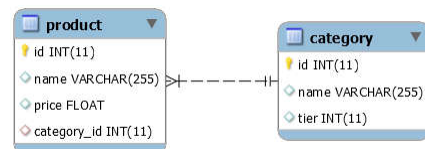
- Where (operação "filter" / "restrição")
- Select (operação "map" / "projeção")
- OrderBy, OrderByDescending, ThenBy, ThenByDescending
- Skip, Take
- First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault
- Max, Min, Count, Sum, Average, Aggregate (operação "reduce")
- GroupBy

<https://github.com/acenelio/linq-demo1>

# Nivelamento: Álgebra Relacional e SQL

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves



CATEGORY

ID	NAME	TIER
1	Tools	2
2	Computers	1
3	Electronics	1

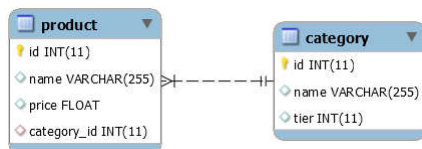
PRODUCT

ID	NAME	PRICE	CATEGORY_ID
1	Computer	1100.0	2
2	Hammer	90.0	1
3	TV	1700.0	3
4	Notebook	1300.0	2



# Operações básicas da álgebra relacional

- Restrição
- Projeção
- Produto cartesiano
- Junção (produto cartesiano + restrição de chaves correspondentes)



CATEGORY

ID	NAME	TIER
1	Tools	2
2	Computers	1
3	Electronics	1

PRODUCT

ID	NAME	PRICE	CATEGORY_ID
1	Computer	1100.0	2
2	Hammer	90.0	1
3	TV	1700.0	3
4	Notebook	1300.0	2

## Operação "produto cartesiano":

```
SELECT *
FROM PRODUCT, CATEGORY
```

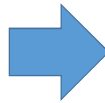
ID	NAME	PRICE	CATEGORY_ID	ID	NAME	TIER
1	Computer	1100.0	2	1	Tools	2
2	Hammer	90.0	1	1	Tools	2
3	TV	1700.0	3	1	Tools	2
4	Notebook	1300.0	2	1	Tools	2
1	Computer	1100.0	2	2	Computers	1
2	Hammer	90.0	1	2	Computers	1
3	TV	1700.0	3	2	Computers	1
4	Notebook	1300.0	2	2	Computers	1
1	Computer	1100.0	2	3	Electronics	1
2	Hammer	90.0	1	3	Electronics	1
3	TV	1700.0	3	3	Electronics	1
4	Notebook	1300.0	2	3	Electronics	1

## Operação "junção":

```
SELECT *
FROM PRODUCT, CATEGORY
WHERE
    PRODUCT.CATEGORY_ID = CATEGORY.ID
```

```
SELECT *
FROM PRODUCT
INNER JOIN CATEGORY cat
ON PRODUCT.CATEGORY_ID = cat.ID
```

ID	NAME	PRICE	CATEGORY_ID	ID	NAME	TIER
1	Computer	1100.0	2	1	Tools	2
2	Hammer	90.0	1	1	Tools	2
3	TV	1700.0	3	1	Tools	2
4	Notebook	1300.0	2	1	Tools	2
1	Computer	1100.0	2	2	Computers	1
2	Hammer	90.0	1	2	Computers	1
3	TV	1700.0	3	2	Computers	1
4	Notebook	1300.0	2	2	Computers	1
1	Computer	1100.0	2	3	Electronics	1
2	Hammer	90.0	1	3	Electronics	1
3	TV	1700.0	3	3	Electronics	1
4	Notebook	1300.0	2	3	Electronics	1

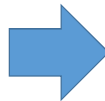


ID	NAME	PRICE	CATEGORY_ID	ID	NAME	TIER
2	Hammer	90.0	1	1	Tools	2
1	Computer	1100.0	2	2	Computers	1
4	Notebook	1300.0	2	2	Computers	1
3	TV	1700.0	3	3	Electronics	1

## Operação "restrição":

```
SELECT *
FROM PRODUCT
INNER JOIN CATEGORY cat ON PRODUCT.CATEGORY_ID = cat.ID
WHERE CATEGORY.NAME = 'Computers'
```

ID	NAME	PRICE	CATEGORY_ID	ID	NAME	TIER
1	Computer	1100.0	2	1	Tools	2
2	Hammer	90.0	1	1	Tools	2
3	TV	1700.0	3	1	Tools	2
4	Notebook	1300.0	2	1	Tools	2
1	Computer	1100.0	2	2	Computers	1
2	Hammer	90.0	1	2	Computers	1
3	TV	1700.0	3	2	Computers	1
4	Notebook	1300.0	2	2	Computers	1
1	Computer	1100.0	2	3	Electronics	1
2	Hammer	90.0	1	3	Electronics	1
3	TV	1700.0	3	3	Electronics	1
4	Notebook	1300.0	2	3	Electronics	1



ID	NAME	PRICE	CATEGORY_ID	ID	NAME	TIER
2	Hammer	90.0	1	1	Tools	2
1	Computer	1100.0	2	2	Computers	1
4	Notebook	1300.0	2	2	Computers	1
3	TV	1700.0	3	3	Electronics	1

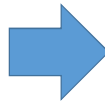


ID	NAME	PRICE	CATEGORY_ID	ID	NAME	TIER
1	Computer	1100.0	2	2	Computers	1
4	Notebook	1300.0	2	2	Computers	1

## Operação "projeção":

```
SELECT PRODUCT.*, CATEGORY.NAME  
FROM PRODUCT  
INNER JOIN CATEGORY cat ON PRODUCT.CATEGORY_ID = cat.ID  
WHERE CATEGORY.NAME = 'Computers'
```

ID	NAME	PRICE	CATEGORY_ID	ID	NAME	TIER
1	Computer	1100.0	2	1	Tools	2
2	Hammer	90.0	1	1	Tools	2
3	TV	1700.0	3	1	Tools	2
4	Notebook	1300.0	2	1	Tools	2
1	Computer	1100.0	2	2	Computers	1
2	Hammer	90.0	1	2	Computers	1
3	TV	1700.0	3	2	Computers	1
4	Notebook	1300.0	2	2	Computers	1
1	Computer	1100.0	2	3	Electronics	1
2	Hammer	90.0	1	3	Electronics	1
3	TV	1700.0	3	3	Electronics	1
4	Notebook	1300.0	2	3	Electronics	1



ID	NAME	PRICE	CATEGORY_ID	ID	NAME	TIER
2	Hammer	90.0	1	1	Tools	2
1	Computer	1100.0	2	2	Computers	1
4	Notebook	1300.0	2	2	Computers	1
3	TV	1700.0	3	3	Electronics	1



ID	NAME	PRICE	CATEGORY_ID	ID	NAME	TIER
1	Computer	1100.0	2	2	Computers	1
4	Notebook	1300.0	2	2	Computers	1



ID	NAME	PRICE	CATEGORY_ID	NAME
1	Computer	1100.0	2	Computers
4	Notebook	1300.0	2	Computers

# Demo: LINQ com notação similar a SQL

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

## Código fonte

- <https://github.com/acenelio/linq-demo2>

## Exercício resolvido

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

Fazer um programa para ler um conjunto de produtos a partir de um arquivo em formato .csv (suponha que exista pelo menos um produto).

Em seguida mostrar o preço médio dos produtos. Depois, mostrar os nomes, em ordem decrescente, dos produtos que possuem preço inferior ao preço médio.

Veja exemplo na próxima página.

<https://github.com/acenelio/lambda6-csharp>

#### **Input file:**

```
Tv,900.00  
Mouse,50.00  
Tablet,350.50  
HD Case,80.90  
Computer,850.00  
Monitor,290.00
```

#### **Execution:**

```
Enter full file path: c:\temp\in.txt  
Average price: 420.23  
Tablet  
Mouse  
Monitor  
HD Case
```

<https://github.com/acenelio/lambda6-csharp>

# Exercício de fixação

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

Fazer um programa para ler os dados (nome, email e salário) de funcionários a partir de um arquivo em formato .csv.

Em seguida mostrar, em ordem alfabética, o email dos funcionários cujo salário seja superior a um dado valor fornecido pelo usuário.

Mostrar também a soma dos salários dos funcionários cujo nome começa com a letra 'M'.

Veja exemplo na próxima página.

Employee
- name : String
- email : String
- salary : Double

<https://github.com/acenelio/lambda7-csharp>

### Input file:

```
Maria,maria@gmail.com,3200.00  
Alex,alex@gmail.com,1900.00  
Marco,marco@gmail.com,1700.00  
Bob,bob@gmail.com,3500.00  
Anna,anna@gmail.com,2800.00
```

### Execution:

```
Enter full file path: c:\temp\in.txt  
Enter salary: 2000.00  
Email of people whose salary is more than 2000.00:  
anna@gmail.com  
bob@gmail.com  
maria@gmail.com  
Sum of salary of people whose name starts with 'M': 4900.00
```

<https://github.com/acenelio/lambda7-csharp>