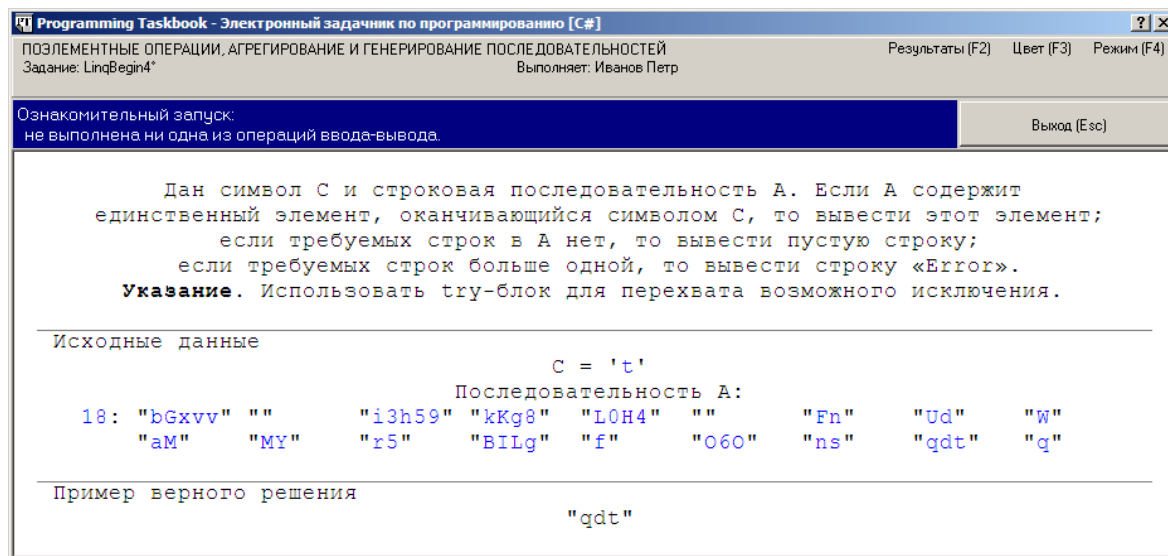


Поэлементные операции: LinqBegin4



```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using PT4;

namespace PT4Tasks
{
    public class MyTask : PT
    {
        // При решении задач группы LinqBegin доступны следующие
        // дополнительные методы, определенные в задачнике:
        //
        // GetEnumerableInt() - ввод числовой последовательности;
        //
        // GetEnumerableString() - ввод строковой последовательности;
        //
        // Put() (метод расширения) - вывод последовательности;
        //
        // Show() и Show(cmt) (методы расширения) - отладочная печать
        // последовательности, cmt - строковый комментарий;
        //
        // Show(e => r) и Show(cmt, e => r) (методы расширения) -
        // отладочная печать значений r, полученных из элементов e
        // последовательности, cmt - строковый комментарий.

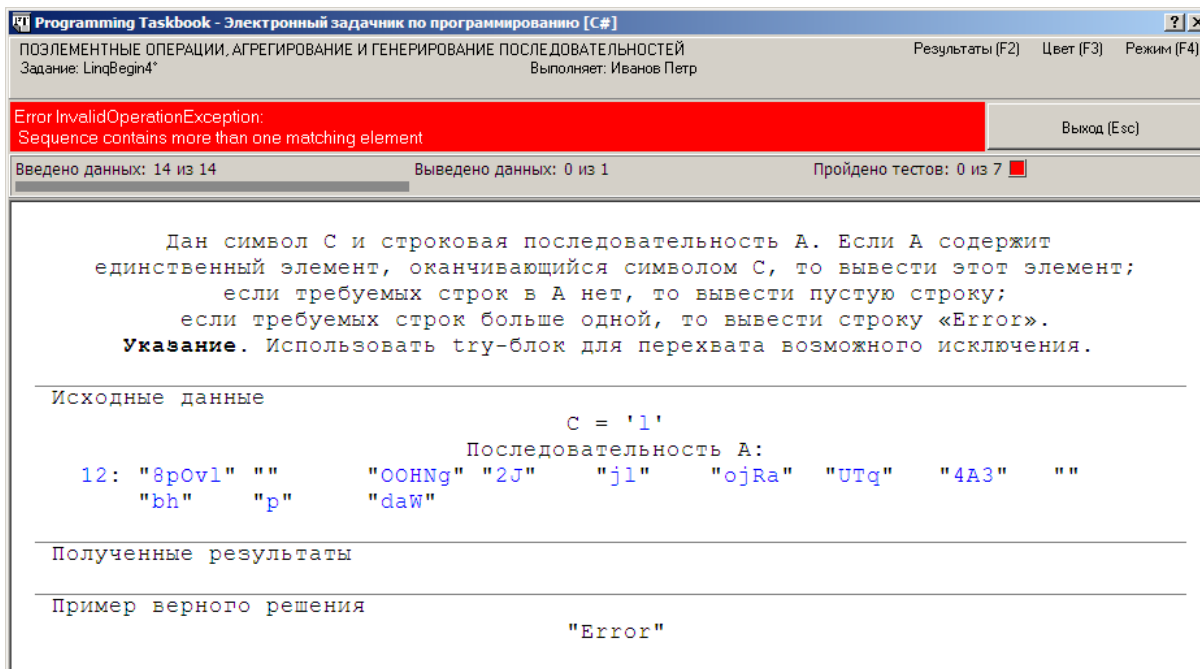
        public static void Solve()
        {
            Task("LinqBegin4");
        }
    }
}

```

```

public static void Solve()
{
    Task("LinqBegin4");
    char c = GetChar();
    var a = GetEnumerableString();
    Put(a.Single(e => e.Length != 0 && e[e.Length - 1] == c));
}

```



```

public static void Solve()
{
    Task("LinqBegin4");
    char c = GetChar();
    var a = GetEnumerableString();
    try
    {
        Put(a.Single(e => e.Length != 0 && e[e.Length - 1] == c));
    }
    catch (InvalidOperationException ex)
    {
        if (ex.Message.Contains("more"))
            Put("Error");
        else
            Put("");
    }
}

```

```
public static void Solve()
{
    Task("LinqBegin4");
    char c = GetChar();
    var a = GetEnumerableString();
    try
    {
        string r = a.SingleOrDefault(e => e.Length != 0 &&
            e[e.Length - 1] == c);
        Put(r != null ? r : "");
    }
    catch
    {
        Put("Error");
    }
}
```

```
public static void Solve()
{
    Task("LinqBegin4");
    char c = GetChar();
    var a = GetEnumerableString();
    try
    {
        Put(a.SingleOrDefault(e => e.Length != 0 &&
            e[e.Length - 1] == c) ?? "");
    }
    catch
    {
        Put("Error");
    }
}
```

```
public static void Solve()
{
    Task("LinqBegin4");
    char c = GetChar();
    try
    {
        Put(GetEnumerableString().SingleOrDefault(e =>
            e.Length != 0 && e[e.Length - 1] == c) ?? "");
    }
    . . .
}
```

Операция агрегирования и генерирование последовательностей: LinqBegin15

Programming Taskbook - Электронный задачник по программированию [C#]

ПОЗЛЕМЕНТНЫЕ ОПЕРАЦИИ, АГРЕГИРОВАНИЕ И ГЕНЕРИРОВАНИЕ ПОСЛЕДОВАТЕЛЬНОСТЕЙ
Задание: LinqBegin15* Выполняет: Иванов Петр Результаты (F2) Цвет (F3) Режим (F4)

Ознакомительный запуск:
не выполнена ни одна из операций ввода-вывода. Выход (Esc)

Дано целое число N ($0 \leq N \leq 15$). Используя методы Range и Aggregate, найти **факториал** числа N : $N! = 1 \cdot 2 \cdot \dots \cdot N$ при $N \geq 1$; $0! = 1$.
Чтобы избежать целочисленного переполнения, при вычислении факториала использовать вещественный числовой тип.

Исходные данные $N = 5$

Пример верного решения 120.00

```
public static void Solve()
{
    Task("LinqBegin15");
    int n = GetInt();
    double f = Enumerable.Range(1, n).Aggregate(1, (a, e) => a * e);
    Put(f);
}
```

Programming Taskbook - Электронный задачник по программированию [C#]

ПОЗЛЕМЕНТНЫЕ ОПЕРАЦИИ, АГРЕГИРОВАНИЕ И ГЕНЕРИРОВАНИЕ ПОСЛЕДОВАТЕЛЬНОСТЕЙ
Задание: LinqBegin15* Выполняет: Иванов Петр Результаты (F2) Цвет (F3) Режим (F4)

Ошибочное решение. Выход (Esc)

Введено данных: 1 из 1 Выведено данных: 1 из 1 Пройдено тестов: 0 из 7 ■

Дано целое число N ($0 \leq N \leq 15$). Используя методы Range и Aggregate, найти **факториал** числа N : $N! = 1 \cdot 2 \cdot \dots \cdot N$ при $N \geq 1$; $0! = 1$.
Чтобы избежать целочисленного переполнения, при вычислении факториала использовать вещественный числовой тип.

Исходные данные $N = 13$

Полученные результаты 1932053504.00

Пример верного решения 6227020800.00

```
double f = Enumerable.Range(1, n)
    .Aggregate(1, (double a, int e) => a * e);
```

```
double f = Enumerable.Range(1, n).Aggregate(1.0, (a, e) => a * e);
```

```
public static void Solve()
{
    Task("LinqBegin15");
    Put(Enumerable.Range(1, GetInt())
        .Aggregate(1.0, (a, e) => a * e));
}
```

Примечание. При анализе полученного алгоритма может возникнуть желание сократить размер исходной последовательности, генерируемой методом `Range`, удалив из нее первый элемент (со значением 1). Для этого достаточно изменить список параметров метода `Range` следующим образом: `(2, GetInt() - 1)`. Однако при тестировании данного варианта обнаружится, что он неправильно работает для случая $N = 0$, поскольку в этом случае второй параметр принимает недопустимое *отрицательное* значение.

Фильтрация, сортировка, теоретико-множественные операции: *LinqBegin31*

Programming Taskbook - Электронный задачник по программированию [C#]

ФИЛЬТРАЦИЯ, СОРТИРОВКА, ТЕОРЕТИКО-МНОЖЕСТВЕННЫЕ ОПЕРАЦИИ
Задание: LinqBegin31

Выполняет: Иванов Петр

Результаты (F2) Цвет (F3) Режим (F4)

Ознакомительный запуск:
не выполнена ни одна из операций ввода-вывода

Выход (Esc)

Дано целое число $K (>0)$ и последовательность непустых строк A . Строки последовательности содержат только цифры и заглавные буквы латинского алфавита. Найти теоретико-множественное пересечение двух фрагментов A : первый содержит K начальных элементов, а второй – все элементы, расположенные после последнего элемента, оканчивающегося цифрой. Полученную последовательность (не содержащую одинаковых элементов) отсортировать по возрастанию длин строк, а строки одинаковой длины – в лексикографическом порядке по возрастанию.

Исходные данные

$K = 8$

Последовательность A :

10: "G" "ВОНН" "1НА88" "G" "ЗИГА" "G" "G" "ВОНН" "G"
"F"

Пример верного решения

3: "G" "ЗИГА" "ВОНН"

```
int k = GetInt();
var a = GetEnumerableString();
var b = a.Take(k);
var c = a.Reverse()
    .TakeWhile(e => !char.IsDigit(e[e.Length - 1]));
b.Show("B");
c.Show("C");
```

Programming Taskbook - Электронный задачник по программированию [C#]

ФИЛЬТРАЦИЯ, СОРТИРОВКА, ТЕОРЕТИКО-МНОЖЕСТВЕННЫЕ ОПЕРАЦИИ
Задание: LinqBegin31* Выполняет: Иванов Петр

Результаты (F2) Цвет (F3) Режим (F4)

Выведены не все результирующие данные.
Количество выведенных данных: 0 (из 3). Выход (Esc)

Введено данных: 13 из 13 Выведено данных: 0 из 3 Пройдено тестов: 0 из 7

Исходные данные

K = 4

Последовательность A:

11: "D2" "4C" "D2" "JGA" "H" "JGA" "JGA" "5AH" "JGA"
 "4C" "J"

Полученные результаты

:

Пример верного решения

2: "4C" "JGA"

1> В 4: D2 4C D2 JGA
2> С 8: J 4C JGA 5AH JGA JGA H JGA

```
var b = a.Take(k).Show("B");
var c = a.Reverse()
    .TakeWhile(e => !char.IsDigit(e[e.Length - 1])).Show("C");
b.Intersect(c).Show("D").Put();
```

Programming Taskbook - Электронный задачник по программированию [C#]

ФИЛЬТРАЦИЯ, СОРТИРОВКА, ТЕОРЕТИКО-МНОЖЕСТВЕННЫЕ ОПЕРАЦИИ
Задание: LinqBegin31* Выполняет: Иванов Петр

Результаты (F2) Цвет (F3) Режим (F4)

Ошибочное решение. Выход (Esc)

Введено данных: 15 из 15 Выведено данных: 4 из 4 Пройдено тестов: 0 из 7

Исходные данные

K = 5

Последовательность A:

13: "0" "1AJ" "7I" "F" "F" "7I" "F" "F" "D"
 "В" "В" "FE9AH" "I8D6G"

Полученные результаты

3: "1*" "1AJ" "7I" "F"

Пример верного решения

3: "F" "7I" "1AJ"

1> В 5: 0 1AJ 7I F F
2> С 12: I8D6G FE9AH В В D F F 7I F F 7I 1AJ
3> D 3: 1AJ 7I F

```
b.Intersect(c).Show("D")
    .OrderBy(e => e.Length).ThenBy(e => e)
    .Put();
```

```

public static void Solve()
{
    Task("LinqBegin31");
    int k = GetInt();
    var a = GetEnumerableString();
    a.Take(k).Show("B")
        .Intersect(a.Reverse()
            .TakeWhile(e => !char.IsDigit(e[e.Length - 1])).Show("C"))
        .Show("D")
        .OrderBy(e => e.Length).ThenBy(e => e)
        .Put();
}

```

Проецирование: LinqBegin43

ПРОЕЦИРОВАНИЕ
Задание: LinqBegin43

Выполняет: Иванов Петр

Результаты (F2) Цвет (F3) Режим (F4)

Ознакомительный запуск:
не выполнена ни одна из операций ввода-вывода. Выход (Esc)

Дано целое число $K (>0)$ и последовательность непустых строк A . Получить последовательность символов, которая определяется следующим образом: для первых K элементов последовательности A в новую последовательность заносятся символы, стоящие на нечетных позициях данной строки (1, 3, ...), а для остальных элементов A – символы на четных позициях (2, 4, ...). В полученной последовательности поменять порядок элементов на обратный.

Исходные данные

$K = 5$

Последовательность A :

11: "Tu" "VKIxT" "CRDeO" "THT" "oVl" "ExZVM" "SQ" "h" "g"
"Cm5A" "Jp9"

Пример верного решения

17: 'p' 'A' 'm' 'Q' 'V' 'x' 'l' 'o' 'T' 'T' 'O' 'D' 'C' 'T' 'I' 'V' 'T'

Tu VKIXT CRDeO THT oVl ExZVM SQ h g Cm5A Jp9

Для первых K элементов:

```
e.Where((c, i) => i % 2 == 0)
```

Для остальных элементов:

```
e.Where((c, i) => i % 2 == 1)
```

Для элемента с индексом n :

```
e.Where((c, i) => i % 2 == (n < k ? 0 : 1))
```

```
public static void Solve()
{
    Task("LinqBegin43");
    int k = GetInt();
    GetEnumerableString()
        .SelectMany((e, n) =>
            e.Where((c, i) => i % 2 == (n < k ? 0 : 1)).Show())
        .Reverse().Put();
}
```

```
Исходные данные
                                К = 6
                                Последовательность А:
10: "jh"   "r"   "4RNV" "XqQA" "eZdZ" "QgAaq" "CbML" "4x" "fCUud"
    "I"

Полученные результаты
16: 'u' 'C' 'x' 'L' 'b' 'q' 'A' 'Q' 'd' 'e' 'O' 'X' 'N' '4' 'r' 'j'

1> 1: j
2> 1: r
3> 2: 4 N
4> 2: X O
5> 2: e d
6> 3: Q A q
7> 2: b L
8> 1: x
9> 2: C u
10> 0:
```

Примечание. Если метод Show вызывается для последовательности, формируемой в каком-либо лямбда-выражении, то возможна ситуация, когда при выполнении программы окно отладки не появится на экране. Данная ситуация связана с особенностью обработки запросов LINQ, возвращающих последовательности. Подобные запросы являются *отложенными*, т. е. выполняются не в момент их появления в программе, а лишь впоследствии, когда к полученной последовательности будет применена одна из операций, выполняющихся немедленно (например, поэлементная операция, операция агрегирования или операция экспортирования), или когда будет организован цикл по перебору элементов этой последовательности (подобные циклы организуются, в частности, в методах Show и Put). Если ни одно из указанных действий в программе не выполняется, то не будут выполнены и отложенные запросы, а значит, и входящие в их лямбда-выражения методы Show. Для моделирования подобной ситуации достаточно удалить в предыдущей программе вызов метода Put.


```

public static void Solve()
{
    Task("LinqBegin43");
    int k = GetInt();
    var a = GetEnumerableString();
    a.Take(k).SelectMany(e => e.Where((c, i) => i % 2 == 0))
        .Concat(a.Skip(k)
            .SelectMany(e => e.Where((c, i) => i % 2 == 1)))
        .Reverse().Put();
}

```

Объединение: LinqBegin52, LinqBegin54

ОБЪЕДИНЕНИЕ И ГРУППИРОВКА
Задание: LinqBegin52* Выполняет: Иванов Петр Результаты (F2) Цвет (F3) Режим (F4)

Ознакомительный запуск:
не выполнена ни одна из операций ввода-вывода. Вывод (Esc)

Даны строковые последовательности A и B; все строки в каждой последовательности различны, имеют ненулевую длину и содержат только цифры и заглавные буквы латинского алфавита. Получить последовательность всевозможных комбинаций вида «E_A=E_B», где E_A – некоторый элемент из A, E_B – некоторый элемент из B, причем оба элемента оканчиваются цифрой (например, «AF3=D78»). Упорядочить полученную последовательность в лексикографическом порядке по возрастанию элементов E_A, а при одинаковых элементах E_A – в лексикографическом порядке по убыванию элементов E_B.
Указание. Для перебора комбинаций использовать методы SelectMany и Select.

Исходные данные

15:	"AJ"	"HE9"	"0"	Последовательность A:	"J2"	"B7"	"DBH"	"D"	"CHG"	"2"
	"D9C"	"4"	"1"		"5DD"	"B"	"2A7"			
				Последовательность B:	"7IB"	"E"	"F"	"J"	"2"	"0"
10:	"EJ"	"1"	"BG"							
	"1I9"									

Пример верного решения

32:	"0=2"	"0=1I9"	"0=1"	"0=0"	"1=2"	"1=1I9"	"1=1"
	"1=0"	"2=2"	"2=1I9"	"2=1"	"2=0"	"2A7=2"	"2A7=1I9"
	"2A7=1"	"2A7=0"	"4=2"	"4=1I9"	"4=1"	"4=0"	"B7=2"
	"B7=1I9"	"B7=1"	"B7=0"	"HE9=2"	"HE9=1I9"	"HE9=1"	"HE9=0"
	"J2=2"	"J2=1I9"	"J2=1"	"J2=0"			

```

var a = GetEnumerableString()
    .Where(e => char.IsDigit(e[e.Length - 1])).Show("A");
var b = GetEnumerableString()
    .Where(e => char.IsDigit(e[e.Length - 1])).Show("B");

```

Для доступа к последнему символу строки e вместо выражения e[e.Length - 1] можно использовать запрос LINQ e.Last() (при этом строка e будет интерпретироваться как последовательность символов).

```

a.SelectMany(e1 => b.Select(e2 => e1 + "=" + e2)).Put();

```

Programming Taskbook - Электронный задачник по программированию [C#] ? X

ОБЪЕДИНЕНИЕ И ГРУППИРОВКА
Задание: LinqBegin52* Выполняет: Иванов Петр Результаты (F2) Цвет (F3) Режим (F4)

Ошибочное решение. Выход (Esc)

Введено данных: 28 из 28 Выведено данных: 10 из 10 Пройдено тестов: 0 из 7

Исходные данные

Последовательность A:
10: "FGF" "3A5" "3C" "H" "E" "8" "J" "A" "3B6"
"C"

Последовательность B:
16: "23" "C" "HFG" "FF" "E9B" "57C" "8C" "7" "60D"
"89" "CII" "J" "3A" "30I" "E2C" "EBB"

Полученные результаты

9: "3A5=23" "3A5=7" "3A5=89" "8=23" "8=7" "8=89" "3B6="*
"3B6="* "3B6="*

Пример верного решения

9: "3A5=89" "3A5=7" "3A5=23" "3B6=89" "3B6=7" "3B6=23" "8=89"
"8=7" "8=23"

1> A 3: 3A5 8 3B6
2> B 3: 23 7 89

```
a.SelectMany(e1 => b.Select(e2 => e1 + "=" + e2))
  .OrderBy(e => e.Split('=')[0])
  .ThenByDescending(e => e.Split('=')[1])
  .Put();
```

```
public static void Solve()
{
    Task("LinqBegin52");
    var a = GetEnumerableString()
        .Where(e => char.IsDigit(e[e.Length - 1]))
        .OrderBy(e => e).Show("A");
    var b = GetEnumerableString()
        .Where(e => char.IsDigit(e[e.Length - 1]))
        .OrderByDescending(e => e).Show("B");
    a.SelectMany(e1 => b.Select(e2 => e1 + "=" + e2)).Put();
}
```

Programming Taskbook - Электронный задачник по программированию [C#]

Объединение и группировка
Задание: LinqBegin52

Выполняет: Иванов Петр

Результаты (F2) Цвет (F3) Режим (F4)

Задание выполнено! Выход (Esc)

Введено данных: 25 из 25 Выведено данных: 31 из 31 Пройдено тестов: 7 из 7

Исходные данные

13:	"2I"	"0"	"C"	"2"	"IDH"	"4HF"	"B86"	"I"	"1DC"
	"49"	"9G"	"283"	"6C"					
10:	"5"	"4"	"47D"	"JG"	"1"	"E"	"F"	"6"	"5B8"
	"3"								

Полученные результаты

30:	"0=6"	"0=5B8"	"0=5"	"0=4"	"0=3"	"0=1"	"2=6"
	"2=5B8"	"2=5"	"2=4"	"2=3"	"2=1"	"283=6"	"283=5B8"
	"283=5"	"283=4"	"283=3"	"283=1"	"49=6"	"49=5B8"	"49=5"
	"49=4"	"49=3"	"49=1"	"B86=6"	"B86=5B8"	"B86=5"	"B86=4"
	"B86=3"	"B86=1"					

1> A 5: 0 2 283 49 B86
2> B 6: 6 5B8 5 4 3 1

Выражения запросов

```
a.SelectMany(e1 => b.Select(e2 => e1 + "=" + e2)).Put();
```

```
var c =
    from e1 in a
    from e2 in b
    select e1 + "=" + e2;
c.Put();
```

```
(from e1 in a
 from e2 in b
 select e1 + "=" + e2).Put();
```

```
public static void Solve()
{
    Task("LinqBegin52");
    var a = GetEnumerableString()
        .Where(e => char.IsDigit(e[e.Length - 1]));
    var b = GetEnumerableString()
        .Where(e => char.IsDigit(e[e.Length - 1]));
    (from e1 in a
     from e2 in b
     orderby e1, e2 descending
     select e1 + "=" + e2).Put();
}
```

```
public static void Solve()
{
    Task("LinqBegin52");
    var a = GetEnumerableString();
    var b = GetEnumerableString();
    (from e1 in a
     where char.IsDigit(e1.Last())
     from e2 in b
     where char.IsDigit(e2.Last())
     orderby e1, e2 descending
     select e1 + "=" + e2).Put();
}
```

Может возникнуть впечатление, что второй вариант решения (использующий выражение запроса) выполняет *больше* действий, чем первый вариант (основанный на вызовах методов LINQ). В самом деле, в первом варианте отбор строк оканчивающихся цифрой, выполняется при построении последовательностей *a* и *b*, т. е., *казалось бы*, один раз для каждой последовательности, а во втором варианте этот отбор выполняется в процессе построения перекрестного объединения, при котором обращение ко второй последовательности производится многократно (а значит, многократно проводится и отбор ее элементов).

На самом деле многократный отбор элементов для последовательности *b* будет проводиться и в первом варианте решения. Это связано с *отложенным* характером выполнения запросов, возвращающих последовательности. При выполнении оператора

```
var b = GetEnumerableString()
    .Where(e => char.IsDigit(e[e.Length - 1]))
    .OrderByDescending(e => e).Show("B");
```

в переменную *b* записывается не результат выполнения методов *Where* и *OrderByDescending*, а *сами эти запросы* (в специальном формате). Немедленно выполняются только вспомогательные методы *GetEnumerableString* и *Show*, реализованные в задачнике. При последующих обращениях к последовательности *b*, при которых требуется организовать перебор всех ее элементов, методы *Where* и *OrderByDescending* каждый раз *выполняются заново*.

```
var b = GetEnumerableString()
    .Where(e => char.IsDigit(e.Show("W").Last()))
    .OrderByDescending(e => e.Show("O"));
```

В данной модификации при каждом вызове лямбда-выражения для методов *Where* и *OrderByDescending* в раздел отладки записывается аргумент этого лямбда-выражения (в виде последовательности символов), причем аргумент для лямбда-выражения метода *Where* снабжается комментарием «W», а аргумент для лямбда-выражения метода *OrderByDescending* — комментарием «O». При запуске программы в разделе отладки будут выведены все элементы исходной последовательности *B*, снабженные комментарием «W», а затем — все оканчивающиеся цифрой элементы этой же последовательно-

сти, снабженные комментарием «О», причем *описанный набор данных будет выведен столько раз, сколько имеется элементов исходной последовательности A, оканчивающихся цифрой*. Это показывает, что отбор элементов последовательности B и их сортировка выполняются при обработке *каждого* элемента последовательности A, используемого при построении перекрестного объединения.

```
public static void Solve()
{
    Task("LinqBegin52");
    var a = GetEnumerableString()
        .Where(e => char.IsDigit(e[e.Length - 1]))
        .OrderBy(e => e).Show("A").ToArray();
    var b = GetEnumerableString()
        .Where(e => char.IsDigit(e[e.Length - 1]))
        .OrderByDescending(e => e).Show("B").ToArray();
    a.SelectMany(e1 => b.Select(e2 => e1 + "=" + e2)).Put();
}
```

Если модифицировать данную программу тем же способом, которым был модифицирован предыдущий вариант программы (добавив в лямбда-выражения методов Where и OrderByDescending для последовательности b вызовы Show), то элементы последовательности b, обрабатываемые методами Where и OrderByDescending, будут выведены на экран лишь по одному разу.

Построение плоского левого внешнего объединения: LinqBegin54

Programming Taskbook - Электронный задачник по программированию [C#]

ОБЪЕДИНЕНИЕ И ГРУППИРОВКА Выполняет: Иванов Петр Результаты (F2) Цвет (F3) Режим (F4)

Задание: LinqBegin54*

Ознакомительный запуск:
не выполнена ни одна из операций ввода-вывода. Выход (Esc)

Даны строковые последовательности A и B; все строки в каждой последовательности различны, имеют ненулевую длину и содержат только цифры и заглавные буквы латинского алфавита. Найти последовательность всех пар строк, удовлетворяющих следующим условиям: первый элемент пары принадлежит последовательности A, а второй либо является одним из элементов последовательности B, начинающихся с того же символа, что и первый элемент пары, либо является пустой строкой (если B не содержит ни одной подходящей строки). Результирующая последовательность называется **левым внешним объединением** последовательностей A и B по **ключу**, определяемому первыми символами исходных строк. Представить найденное объединение в виде последовательности строк вида «E_A.E_B», где E_A – элемент из A, а E_B – либо один из соответствующих ему элементов из B, либо пустая строка. Расположить элементы полученной строковой последовательности в лексикографическом порядке по возрастанию.

Указание. Использовать методы GroupJoin, DefaultIfEmpty, Select и SelectMany.

Исходные данные

10:	"7"	"7BI"	"FC"	Последовательность A:					
	"I1J"			"I"	"EA"	"AC"	"E4"	"G8I"	"J"
11:	"J"	"9"	"67"	Последовательность B:					
	"64H"	"GI5"		"I"	"980"	"J64"	"7"	"C"	"5EJ"

Пример верного решения

11:	"7.7"	"7BI.7"	"AC."	"E4."	"EA."	"FC."	"G8I.GI5"
	"I.I"	"I1J.I"	"J.J"	"J.J64"			

```

public static void Solve()
{
    Task("LinqBegin54");
    GetEnumerableString().GroupJoin(GetEnumerableString(),
        e => e[0], e => e[0],
        (e1, ee2) => ee2.Select(e => e1 + "." + e))
        .SelectMany(e => e).OrderBy(e => e).Show().Put();
}

```

Programming Taskbook - Электронный задачник по программированию [C#]

ОБЪЕДИНЕНИЕ И ГРУППИРОВКА
Задание: LinqBegin54

Выполняет: Иванов Петр

Результаты (F2) Цвет (F3) Режим (F4)

Выведены не все результирующие данные.
Количество выведенных данных: 8 (из 14)

Вывод (Esc)

Введено данных: 24 из 24 Выведено данных: 8 из 14 Пройдено тестов: 0 из 7

Исходные данные

12: "В" "I" "G" "DJC" "2" "ICF" "E7" "CAB" "HE"
"8" "AD2" "FI2"

Последовательность A:

10: "EA" "8" "74I" "J8B" "C" "HGH" "G" "AA7" "9"
"GF5"

Последовательность B:

Полученные результаты

7: "8.*" "AD2*" "CAB.C" "E7*" "G.G" "G.GF*" "HE.HG*"

Пример верного решения

13: "2." "8.8" "AD2.AA7" "В." "CAB.C" "DJC." "E7.EA"
"FI2." "G.G" "G.GF5" "HE.HGH" "I." "ICF."

1> 7: 8.8 AD2.AA7 CAB.C E7.EA G.G G.GF5 HE.HGH

```

public static void Solve()
{
    Task("LinqBegin54");
    GetEnumerableString().GroupJoin(GetEnumerableString(),
        e => e[0], e => e[0],
        (e1, ee2) => ee2.DefaultIfEmpty("")
        .Select(e => e1 + "." + e))
        .SelectMany(e => e).OrderBy(e => e).Show().Put();
}

```

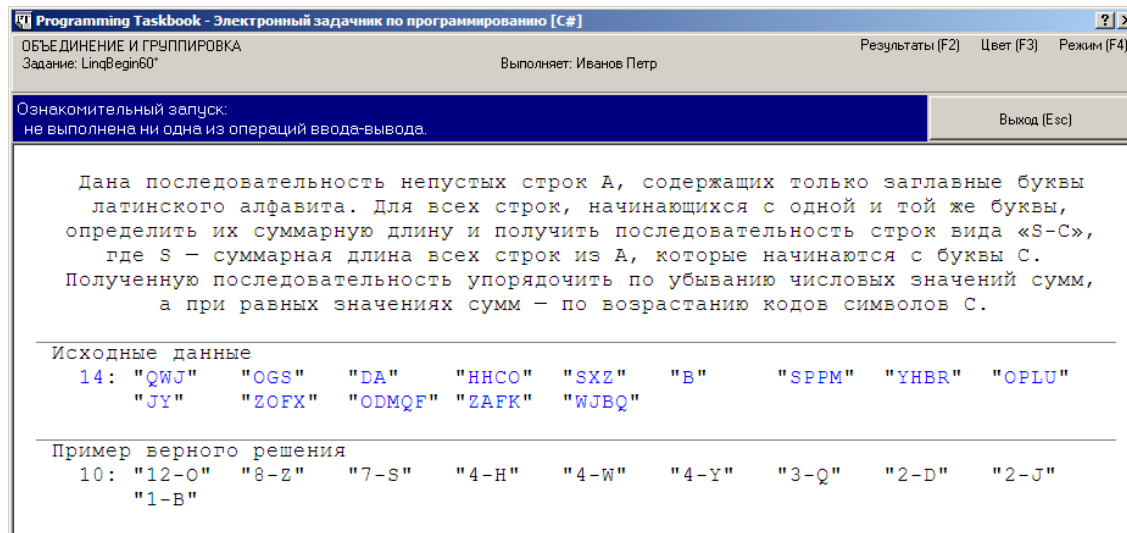
```

public static void Solve()
{
    Task("LinqBegin54");
    (from e1 in GetEnumerableString()
     join e2 in GetEnumerableString()
     on e1[0] equals e2[0]
     into ee2
     from e in ee2.DefaultIfEmpty("")
     orderby e1, e
     select e1 + "." + e).Put();
}

```

```
(from e1 in GetEnumerableString()
 join e2 in GetEnumerableString()
 on e1[0] equals e2[0]
 into ee2
 from e in ee2.DefaultIfEmpty("")
 let r = e1 + "." + e
 orderby r
 select r).Put();
```

Группировка: LinqBegin60



```
GetEnumerableString().GroupBy(e => e[0])
    .OrderByDescending(e => e.Sum(s => s.Length)).ThenBy(e => e.Key)
    .Select(e => e.Sum(s => s.Length) + "-" + e.Key).Put();
```

```
GetEnumerableString().GroupBy(e => e[0], e => e.Length)
    .OrderByDescending(e => e.Sum()).ThenBy(e => e.Key)
    .Select(e => e.Sum() + "-" + e.Key).Put();
```

```
GetEnumerableString()
    .GroupBy(e => e[0], (k, ee) => ee.Sum(s => s.Length) + "-" + k)
    .OrderByDescending(e => int.Parse(e.Split('-')[0]))
    .ThenBy(e => e.Split('-')[1]).Put();
```

```
GetEnumerableString()
    .GroupBy(e => e[0], e => e.Length,
        (k, ee) => ee.Sum() + "-" + k)
    .OrderByDescending(e => int.Parse(e.Split('-')[0]))
    .ThenBy(e => e.Split('-')[1]).Put();
```

```
GetEnumerableString().GroupBy(e => e[0], e => e.Length,
    (k, ee) => new { k, s = ee.Sum() })
    .OrderByDescending(e => e.s).ThenBy(e => e.k)
    .Select(e => e.s + "-" + e.k).Put();
```

```
(from r in
    from e in GetEnumerableString()
    group e.Length by e[0]
let s = r.Sum()
let k = r.Key
orderby s descending, k
select s + "-" + k).Put();
```

```
(from e in GetEnumerableString()
group e.Length by e[0]
into r
let s = r.Sum()
let k = r.Key
orderby s descending, k
select s + "-" + k).Put();
```
