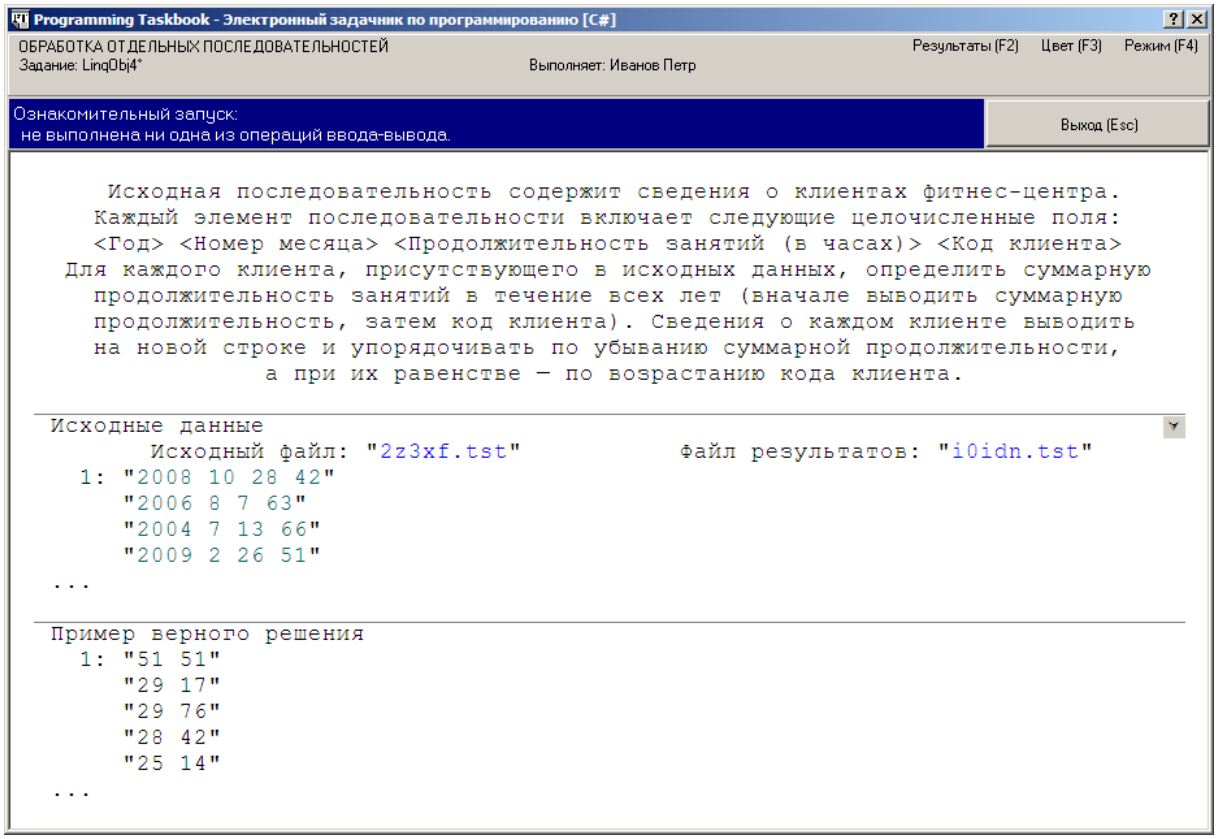
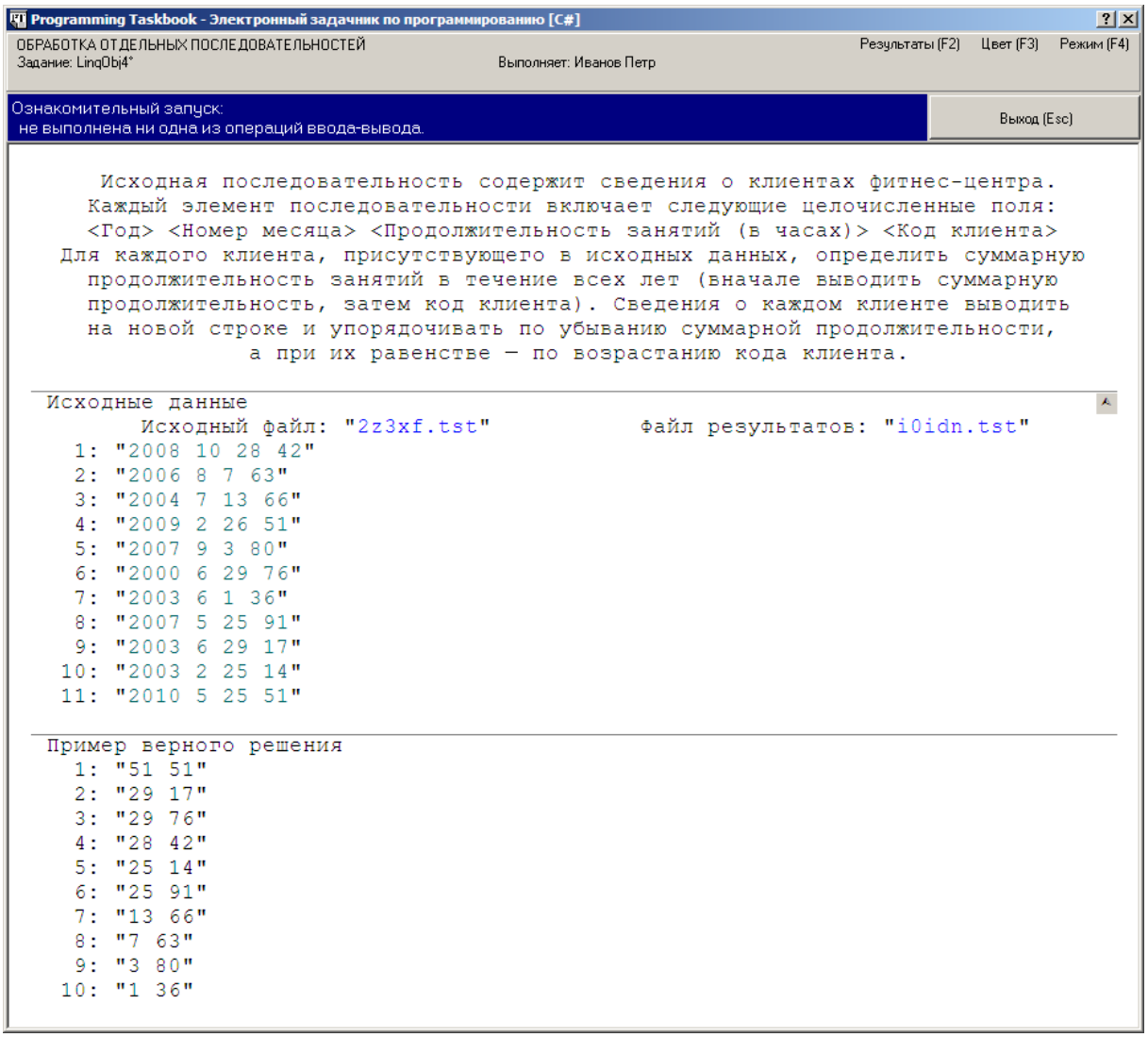
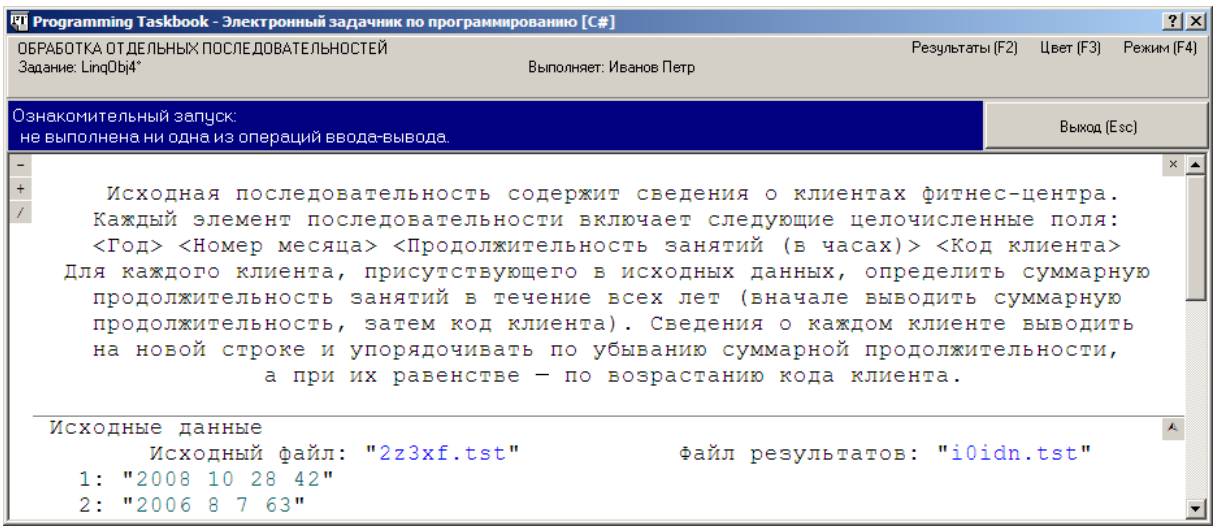
Простое задание на обработку отдельной последовательности: LinqObj4

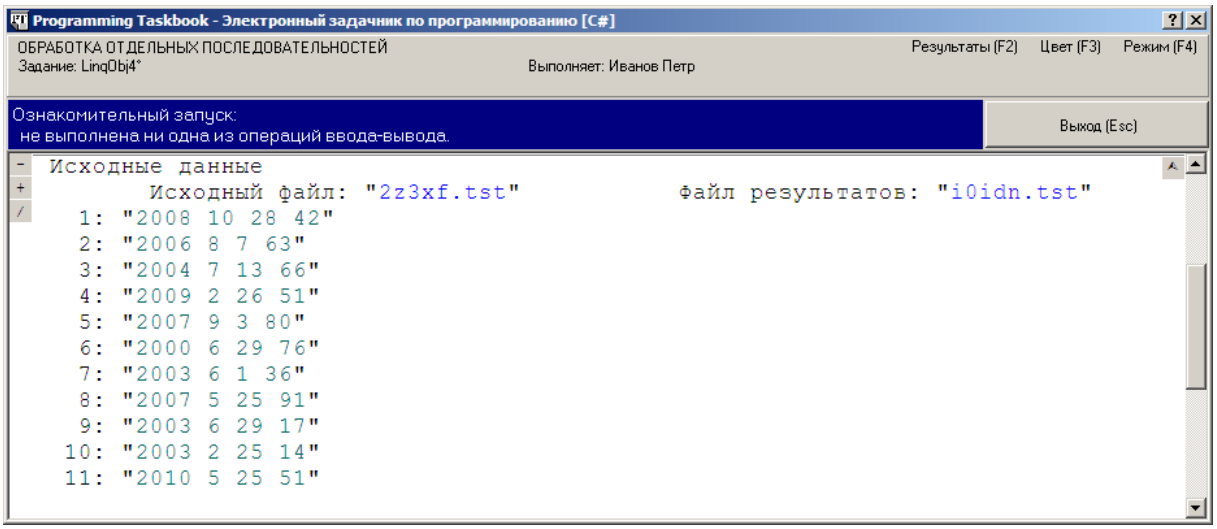



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using PT4;

namespace PT4Tasks
{
  public class MyTask : PT
  {

    // Для чтения набора строк из исходного текстового файла
    // в массив a типа string[] используйте оператор:
    //
    //   a = File.ReadAllLines(GetString(), Encoding.Default);
    //
    // Для записи последовательности r типа IEnumerable<string>
    // в результирующий текстовый файл используйте оператор:
    //
    //   File.WriteAllLines(GetString(), r.ToArray(), Encoding.Default);
    //
    // При решении задач группы LinqObj доступны следующие
    // дополнительные методы расширения, определенные в задачнике:
```

```
//
//   Show() и Show(cmt) - отладочная печать последовательности,
//     cmt - строковый комментарий;
//
//   Show(e => r) и Show(cmt, e => r) - отладочная печать
//     значений r, полученных из элементов e последовательности,
//     cmt - строковый комментарий.

  public static void Solve()
  {
    Task("LinqObj4");

  }
}
. . .
```



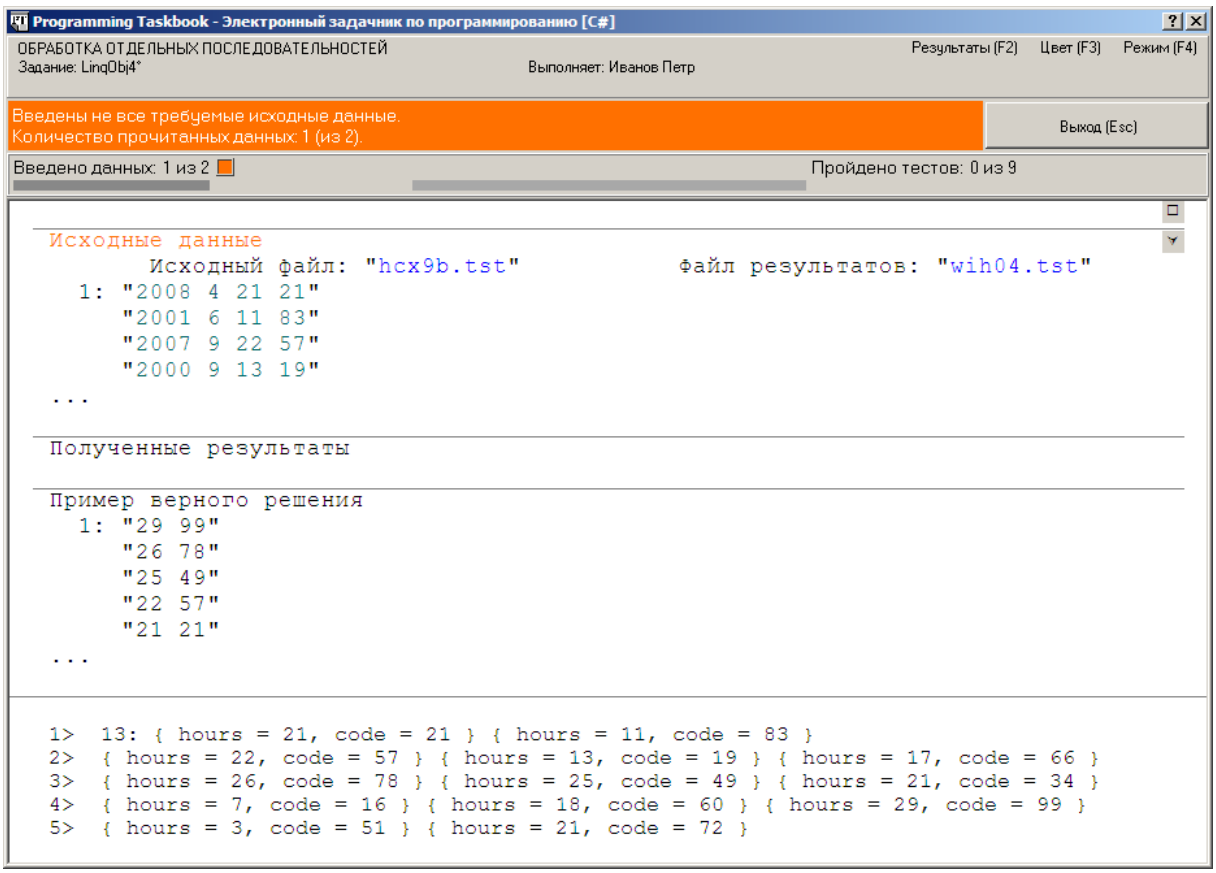





```
File.ReadLines(GetString(), Encoding.Default)
  .Select(e =>
  {
    string[] s = e.Split(' ');
    return new
    {
      hours = int.Parse(s[2]),
      code = int.Parse(s[3])
    };
  })
  .Show();
```

Или:

```
.Select(e => new
{
  hours = int.Parse(e.Split[2]),
  code = int.Parse(e.Split[3])
})
```




```
.GroupBy(e => e.code)
.OrderByDescending(e => e.Sum(c => c.hours))
.ThenBy(e => e.Key)
.Select(e => e.Sum(c => c.hours) + " " + e.Key)

.GroupBy(e => e.code,
  (k, ee) => new {k, sum = ee.Sum(c => c.hours)})
.OrderByDescending(e => e.sum).ThenBy(e => e.k)
.Select(e => e.sum + " " + e.k)

File.WriteAllLines(GetString(), r.ToArray(), Encoding.Default);

public static void Solve()
{
  Task("LinqObj4");
  var r = File.ReadLines(GetString(), Encoding.Default)
    .Select(e =>
    {
      string[] s = e.Split(' ');
      return new
      {
        hours = int.Parse(s[2]),
        code = int.Parse(s[3])
      };
    })
```

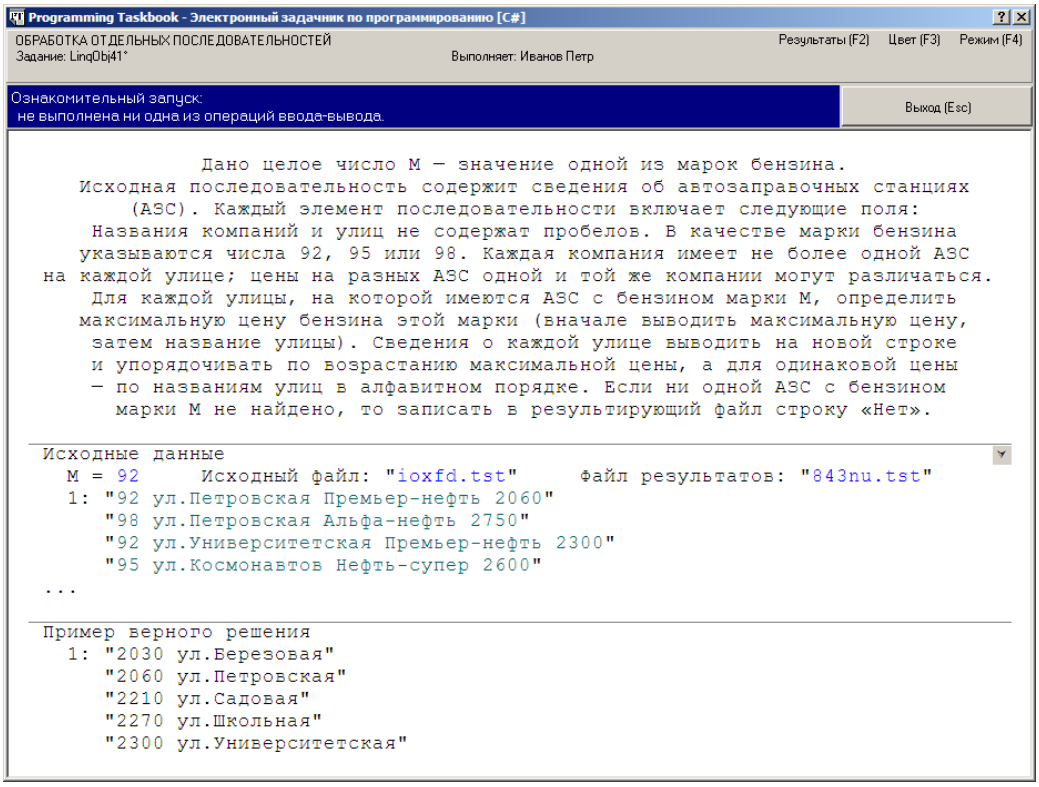
```
    .GroupBy(e => e.code,
      (k, ee) => new { k, sum = ee.Sum(c => c.hours) })
    .OrderByDescending(e => e.sum).ThenBy(e => e.k)
    .Select(e => e.sum + " " + e.k);
  File.WriteAllLines(GetString(), r.ToArray(), Encoding.Default);
}
```

Примечание. В некоторых задачах группы LinqObj в начало файла требуется записать *отдельные* строки (не входящие в последовательность). Чтобы преобразовать строку `s` в одноэлементный массив, достаточно использовать выражение `new string[]{ s }`; напомним, что для объединения последовательностей можно использовать метод Concat.

Вариант решения, использующий выражение запроса:

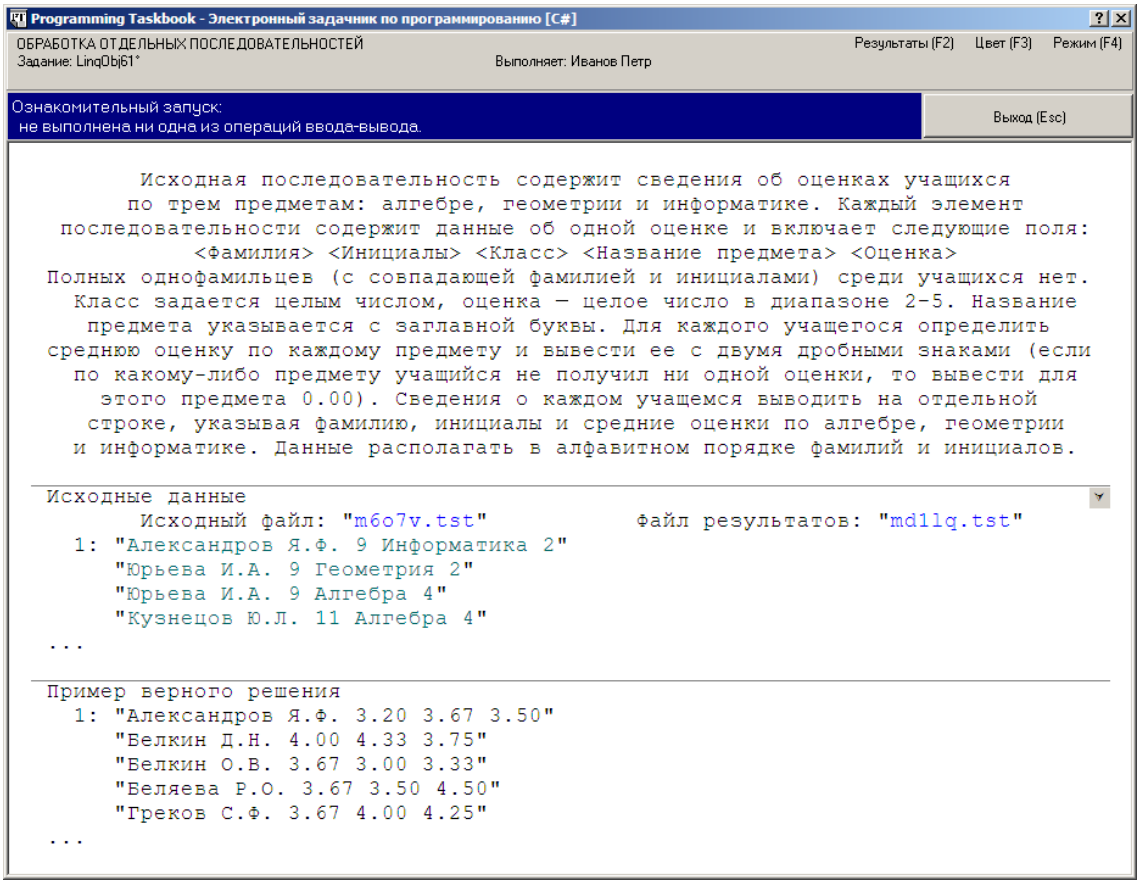
```
public static void Solve()
{
  Task("LinqObj4");
  var r =
    from e in File.ReadLines(GetString(), Encoding.Default)
    let s = e.Split(' ')
    select new
    {
      hours = int.Parse(s[2]),
      code = int.Parse(s[3])
    }
      into e
      group e.hours by e.code
        into e
        let sum = e.Sum()
        orderby sum descending, e.Key
        select sum + " " + e.Key;
  File.WriteAllLines(GetString(), r.ToArray(), Encoding.Default);
}
```

Более сложные задания на обработку отдельных последовательностей: LinqObj41, LinqObj61




```
public static void Solve()
{
  Task("LinqObj41");
  int m = GetInt();
  var r = File.ReadLines(GetString(), Encoding.Default)
    .Select(e =>
    {
      string[] s = e.Split(' ');
      return new
      {
        brand = int.Parse(s[0]),
        street = s[1],
        price = int.Parse(s[3])
      };
    })
    .Where(e => e.brand == m)
    .GroupBy(e => e.street,
      (k, ee) => new { street = k, max = ee.Max(e => e.price) })
    .OrderBy(e => e.max).ThenBy(e => e.street)
    .Select(e => e.max + " " + e.street)
    .DefaultIfEmpty("Нет");
  File.WriteAllLines(GetString(), r.ToArray(), Encoding.Default);
}
```

```
public static void Solve()
{
  Task("LinqObj41");
  int m = GetInt();
  var r =
    (from e in File.ReadLines(GetString(), Encoding.Default)
     let s = e.Split(' ')
     select new
     {
       brand = int.Parse(s[0]),
       street = s[1],
       price = int.Parse(s[3])
     }
       into e
       where e.brand == m
       group e.price by e.street
         into e
         let max = e.Max()
         orderby max, e.Key
         select max + " " + e.Key)
    .DefaultIfEmpty("Нет");
  File.WriteAllLines(GetString(), r.ToArray(), Encoding.Default);
}
```



```
public static void Solve()
{
  Task("LinqObj61");
  string[] subjects = { "Алгебра", "Геометрия", "Информатика" };
  var culture = new System.Globalization.CultureInfo("en-US");
  var r = File.ReadLines(GetString(), Encoding.Default)
    .Select(e =>
    {
      string[] s = e.Split(' ');
      return new
      {
        name = s[0] + " " + s[1],
        subj = s[3],
        mark = int.Parse(s[4])
      };
    })
    .GroupBy(e => e.name, (k, ee) => new
       {
         name = k,
         avrs = subjects
           .GroupJoin(ee, s => s, e => e.subj,
             (s1, ee1) => ee1.Select(e1 => e1.mark)
           .DefaultIfEmpty().Average())
       })
    .OrderBy(e => e.name)
    .Select(e => e.name + e.avrs.Aggregate("",
       (a, d) => a + " " + d.ToString("f2", culture)));
  File.WriteAllLines(GetString(), r.ToArray(), Encoding.Default);
}

public static void Solve()
{
  Task("LinqObj61");
  string[] subjects = { "Алгебра", "Геометрия", "Информатика" };
  var culture = new System.Globalization.CultureInfo("en-US");
  var r =
    from e in File.ReadLines(GetString(), Encoding.Default)
    let s = e.Split(' ')
    select new
    {
      name = s[0] + " " + s[1],
      subj = s[3],
      mark = int.Parse(s[4])
    }
      into e
      group e by e.name
```

```
        into e
        let avrs =
          from e1 in subjects
          join e2 in e
          on e1 equals e2.subj
          into ee
          select
            (from ee1 in ee
             select ee1.mark)
            .DefaultIfEmpty().Average()
        orderby e.Key
        select e.Key + avrs.Aggregate("",
          (a, d) => a + " " + d.ToString("f2", culture));
  File.WriteAllLines(GetString(), r.ToArray(), Encoding.Default);
}
```

Обработка взаимосвязанных последовательностей: LinqObj98

LinqObj98. Даны последовательности *B*, *C*, *D* и *E*, включающие следующие поля:

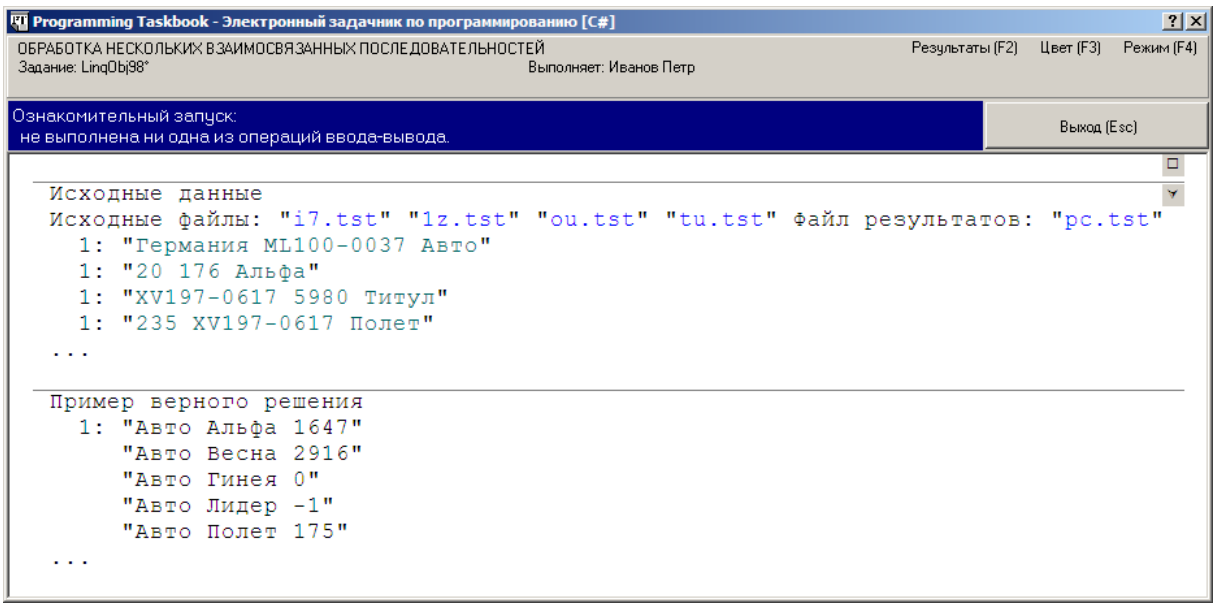
B: <Страна-производитель> <Артикул товара> <Категория>

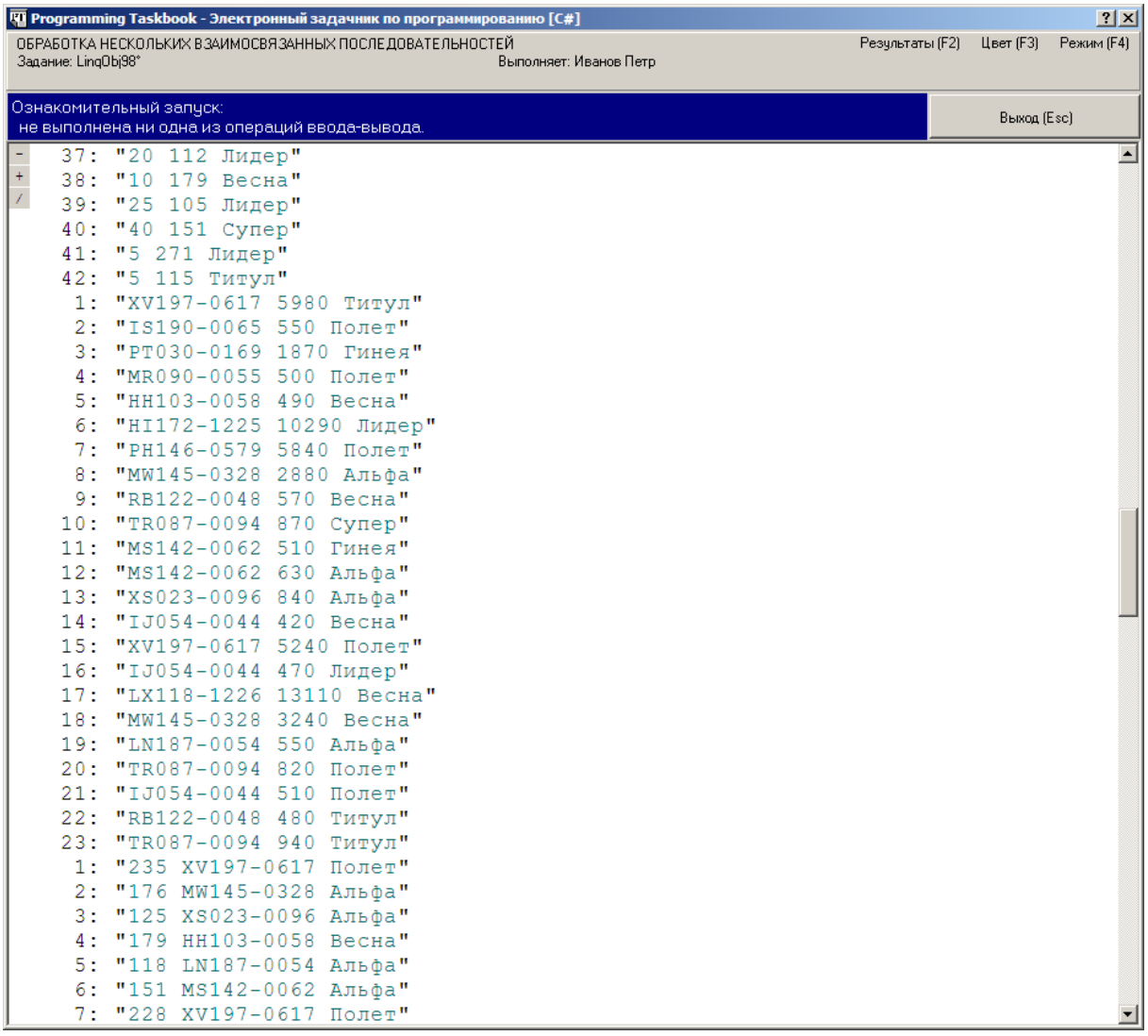
C: <Скидка (в процентах)> <Код потребителя> <Название магазина>

D: <Артикул товара> <Цена (в рублях)> <Название магазина>

E: <Код потребителя> <Артикул товара> <Название магазина>

Свойства последовательностей описаны в преамбуле к данной подгруппе заданий. Для каждой категории товаров и каждого магазина, указанного в *E*, определить суммарный размер скидки на все товары данной категории, проданные в данном магазине (вначале выводится название категории, затем название магазина, затем суммарная скидка). При вычислении размера скидки на товар копейки отбрасываются. Если на проданный товар скидка отсутствует, то ее размер полагается равным 0. Если для некоторой категории товаров в каком-либо магазине не было продаж, то суммарная скидка для этой пары «категория–магазин» полагается равной −1. Сведения о каждой паре «категория–магазин» выводить на новой строке и упорядочивать по названиям категорий в алфавитном порядке, а для одинаковых названий категорий — по названиям магазинов (также в алфавитном порядке).

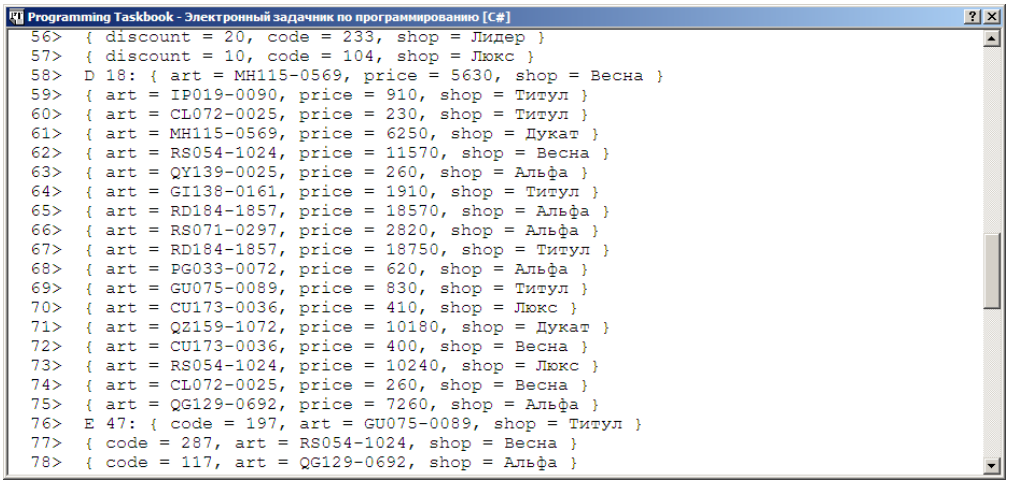




Первый этап решения состоит в формировании последовательностей анонимных типов на основе исходных строковых последовательностей. Для большей наглядности присвоим этим последовательностям имена, которые используются в условии задачи:

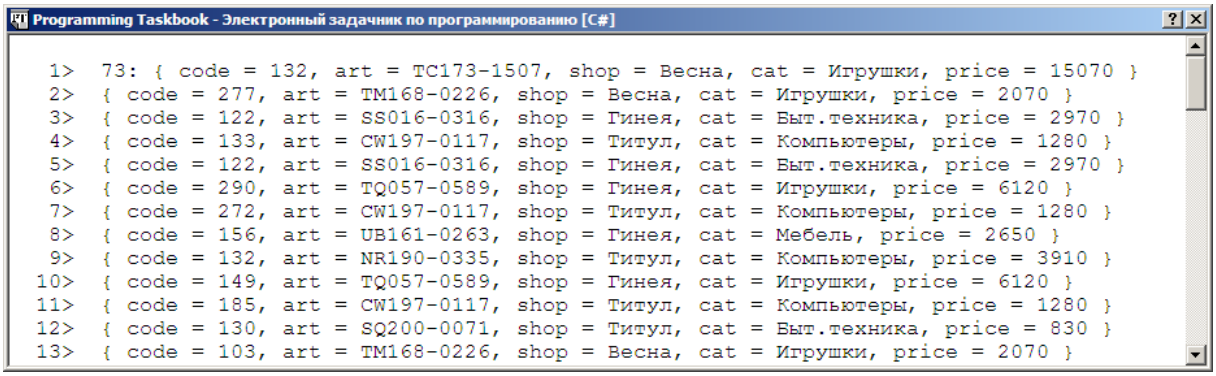
```
var B = File.ReadLines(GetString(), Encoding.Default)
  .Select(e =>
  {
    string[] s = e.Split(' ');
    return new
    {
      art = s[1],
      cat = s[2]
    };
  });
var C = File.ReadLines(GetString(), Encoding.Default)
  .Select(e =>
  {
    string[] s = e.Split(' ');
    return new
    {
      discount = int.Parse(s[0]),
      code = s[1],
      shop = s[2]
    };
  });
var D = File.ReadLines(GetString(), Encoding.Default)
  .Select(e =>
  {
    string[] s = e.Split(' ');
    return new
    {
      art = s[0],
      price = int.Parse(s[1]),
      shop = s[2]
    };
  });
var E = File.ReadLines(GetString(), Encoding.Default)
  .Select(e =>
  {
    string[] s = e.Split(' ');
    return new
    {
      code = s[0],
      art = s[1],
      shop = s[2]
    };
  });
```

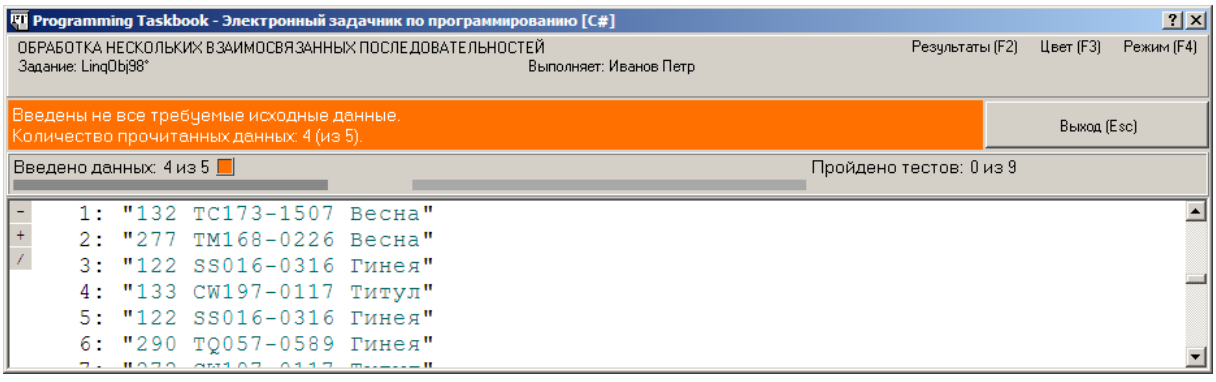
- art — «Артикул товара» (строка),
- cat — «Категория» (строка),
- discount — «Скидка (в процентах)» (целое число),
- code — «Код потребителя» (строка),
- shop — «Название магазина» (строка),
- price — «Цена (в рублях)» (целое число).



На втором этапе решения преобразуем последовательность *E*, добавив в нее дополнительные данные из последовательностей *B*, *C* и *D*.

```
E.Join(B, e1 => e1.art, e2 => e2.art,
  (e1, e2) => new { e1.code, e1.art, e1.shop, e2.cat })
.Join(D, e1 => e1.art + e1.shop, e2 => e2.art + e2.shop,
  (e1, e2) => new { e1.code, e1.art, e1.shop, e1.cat, e2.price })
.Show();
```





Осталось выполнить объединение полученной последовательности и последовательности *C*, в результате которого мы найдем *размер скидки* для каждого проданного товара. Скидка определяется парой «потребитель–магазин»: магазин может предоставлять некоторым потребителям фиксированную скидку *d* (в процентах) на все товары.

В данном случае нельзя применить метод Join, обеспечивающий получение внутреннего объединения, так как для некоторых возможных пар «потребитель–магазин» скидка не определена (т. е. некоторые потребители не имеют скидки в некоторых магазинах). Вместо этого используем метод GroupJoin, возвращающий *левое внешнее объединение*, в котором каждому элементу внешней последовательности (полученной ранее в результате выполненных объединений) сопоставляется *набор* элементов внутренней последовательности *C*. Объединение выполняется по составному ключу — коду потребителя и названию магазина. В результирующем объединении достаточно сохранять три поля: категорию товара, название магазина и размер скидки для данного товара (прочие поля были нужны для построения объединений и вычисления скидки, и в дальнейшем не потребуются):

```
    .GroupJoin(C, e1 => e1.code + e1.shop, e2 => e2.code + e2.shop,
       (e1, ee2) => new {e1.shop, e1.cat,
         discount = ee2.Select(e => e.discount)
           .FirstOrDefault() * e1.price / 100 })
```

Представим в одном выражении все описанные выше операции объединения и сохраним полученную последовательность под именем discounts:

```
var discounts = E.Join(B, e1 => e1.art, e2 => e2.art,
    (e1, e2) => new { e1.code, e1.art, e1.shop, e2.cat })
  .Join(D, e1 => e1.art + e1.shop, e2 => e2.art + e2.shop,
    (e1, e2) => new { e1.code, e1.art, e1.shop, e1.cat, e2.price })
  .GroupJoin(C, e1 => e1.code + e1.shop, e2 => e2.code + e2.shop,
    (e1, ee2) => new
    {
      e1.shop,
      e1.cat,
      discount = ee2.Select(e => e.discount)
        .FirstOrDefault() * e1.price / 100
    });
```

Перейдем к третьему, завершающему этапу решения задачи: обработке полученной последовательности.

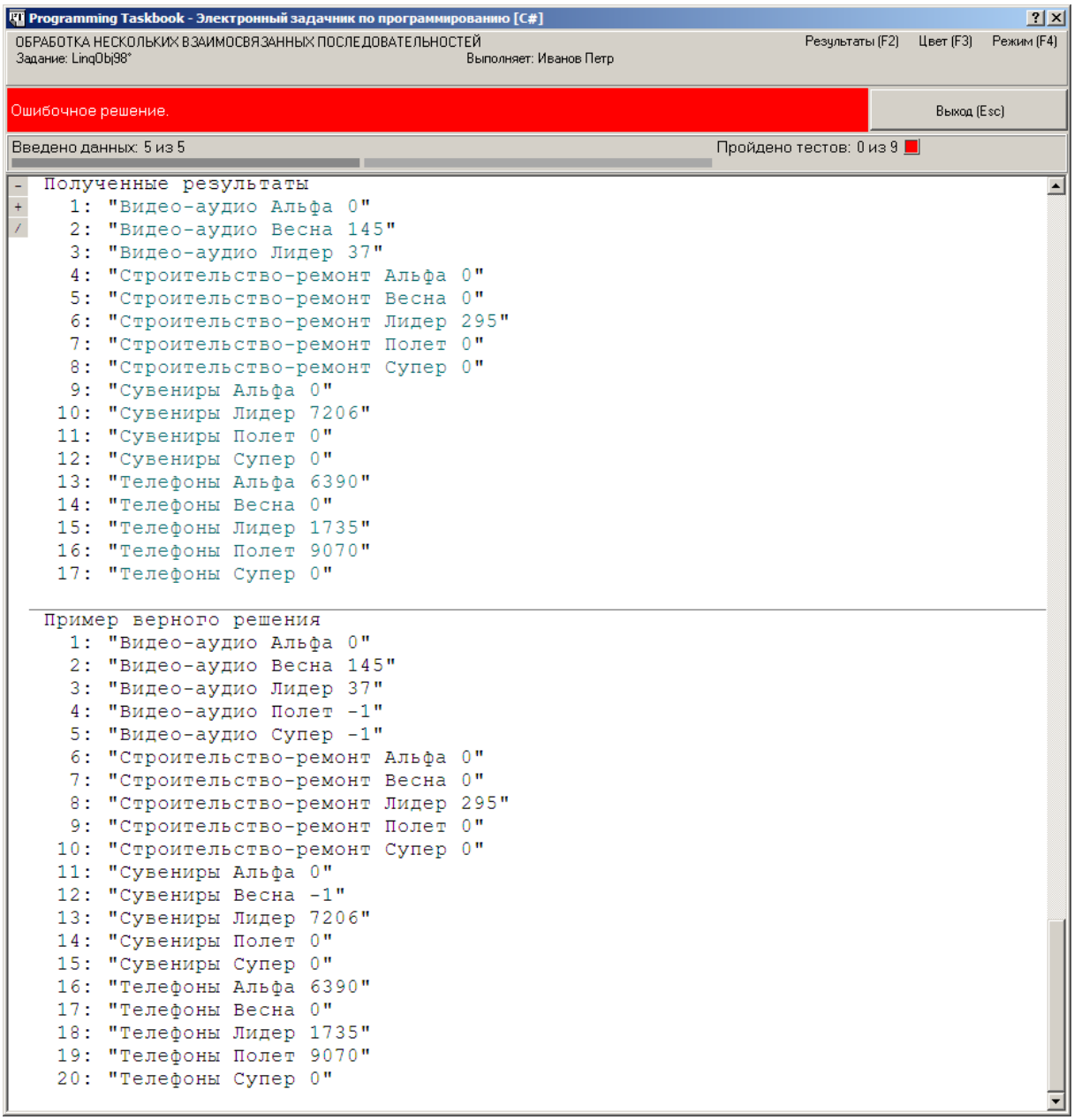
Если бы в задаче требовалось вывести данные только о тех парах «категория–магазин», которые встречаются в последовательности discounts, то было бы достаточно выполнить группировку этой последовательности по составному ключу, включающему поля cat и shop, отсортировать полученную последовательность и найти сумму скидок по всем товарам, имеющим общий ключ:

```
var r = discounts
  .GroupBy(e => e.cat + " " + e.shop, e => e.discount)
  .OrderBy(e => e.Key)
  .Select(e => e.Key + " " + e.Sum());
```

Добавив к решению указанный фрагмент вместе с завершающим оператором

```
File.WriteAllLines(GetString(), r.ToArray(), Encoding.Default);
```

обеспечивающим сохранение последовательности в текстовом файле, и запустив программу, мы получим результат, подобный приведенному на рис. 38. Отличие от правильного решения состоит в том, что в полученной последовательности отсутствует информация о парах «категория–магазин», которым *не соответствует ни одного проданного товара*.



Для того чтобы учесть отсутствующие пары, поступим следующим образом. Вначале сформируем набор cats всех категорий и набор shops всех магазинов, используя группировку последовательности *B* по ключу cat и группировку последовательности *E* по ключу shop (в сгруппированных последовательностях достаточно сохранить только набор ключей группировки):

```
var cats = B.GroupBy(e => e.cat, (k, ee) => k);
var shops = E.GroupBy(e => e.shop, (k, ee) => k);
```

Затем получим перекрестное объединение наборов cats и shops в виде набора строк, используя комбинацию методов SelectMany и Select (см. п. 5.5.2) и отсортируем результат:

```
var cats_shops =
  cats.SelectMany(e1 => shops.Select(e2 => e1 + " " + e2))
    .OrderBy(e => e);
```

И наконец, выполним левое внешнее объединение последовательностей cats_shops и discounts методом GroupJoin, вернув результат в виде набора строк (дополнительная сортировка этого набора уже не потребуется):

```
var r = cats_shops.GroupJoin(discounts, e1 => e1,
  e2 => e2.cat + " " + e2.shop, (e1, ee2) => e1 + " " +
  ee2.Select(e => e.discount).DefaultIfEmpty(-1).Sum());
```

При обработке последовательности ee2 мы учли, что она может оказаться пустой, причем в этом случае следует вернуть значение –1.

Данное решение допускает две модификации: во-первых, при выполнении объединения промежуточной последовательности с последовательностью *D* в результирующей последовательности можно не сохранять поле «Артикул», так как в дальнейшем оно не потребуется; во-вторых, при последующем объединение этой последовательности с последовательностью *C* (методом GroupJoin) можно объединить поля «Категория» и «Магазин», поскольку в дальнейшем они всегда будут использоваться совместно.

Анализируя решение, нетрудно заметить, что все именованные вспомогательные последовательности (discounts, cats, shops, cats_shops) в дальнейшем коде используются по одному разу, и поэтому обращение к ним можно заменить их инициализирующими выражениями. Однако к подобной возможности следует относиться с осторожностью, чтобы не получить излишне громоздкое и поэтому сложное для восприятия выражение. Можно, например, сохранить имена discounts, cats, shops и отказаться от именования перекрестного объединения cats_shops.

```
public static void Solve()
{
  Task("LinqObj98");

  // здесь должен находиться фрагмент, обеспечивающий
  // ввод исходных последовательностей B, C, D и E

  var discounts = E.Join(B, e1 => e1.art, e2 => e2.art,
      (e1, e2) => new { e1.code, e1.art, e1.shop, e2.cat })
    .Join(D, e1 => e1.art + e1.shop, e2 => e2.art + e2.shop,
      (e1, e2) => new { e1.code, e1.shop, e1.cat, e2.price })
    .GroupJoin(C, e1 => e1.code + e1.shop, e2 => e2.code + e2.shop,
      (e1, ee2) => new
      {
        cat_shop = e1.cat + " " + e1.shop,
        discount = ee2.Select(e => e.discount)
          .FirstOrDefault() * e1.price / 100
      });
  var cats = B.GroupBy(e => e.cat, (k, ee) => k);
  var shops = E.GroupBy(e => e.shop, (k, ee) => k);
  var r = cats
    .SelectMany(e1 => shops.Select(e2 => e1 + " " + e2))
    .OrderBy(e => e)
    .GroupJoin(discounts, e1 => e1, e2 => e2.cat_shop,
      (e1, ee2) => e1 + " " +
      ee2.Select(e => e.discount).DefaultIfEmpty(-1).Sum());
  File.WriteAllLines(GetString(), r.ToArray(), Encoding.Default);
}
```

```
public static void Solve()
{
  Task("LinqObj98");

  // здесь должен находиться фрагмент, обеспечивающий
  // ввод исходных последовательностей B, C, D и E

  var r =
    from e1 in
      // нахождение последовательности cats
      from e in B
      group e by e.cat
      // конец вложенного запроса
    from e2 in
      // нахождение последовательности shops
      from e in E
      group e by e.shop
      // конец вложенного запроса
    let s = e1.Key + " " + e2.Key
    orderby s
    select s
      into e1
      join e2 in
         // нахождение последовательности discounts
         from e1 in E
         join e2 in B
         on e1.art equals e2.art
         join e3 in D
         on e1.art + e1.shop equals e3.art + e3.shop
         join e4 in C
         on e1.code + e1.shop equals e4.code + e4.shop
         into ee4
         select new
         {
           cat_shop = e2.cat + " " + e1.shop,
           discount = (from e in ee4 select e.discount)
             .FirstOrDefault() * e3.price / 100
         }
         // конец вложенного запроса
       on e1 equals e2.cat_shop
       into ee2
       select e1 + " " +
         (from e in ee2 select e.discount)
         .DefaultIfEmpty(-1).Sum();
  File.WriteAllLines(GetString(), r.ToArray(), Encoding.Default);
}
```

Завершая обсуждение задачи LinqObj98, приведем фрагмент кода, содержащий выражение запроса для более простой задачи, в которой требуется вывести информацию только о тех парах «категория–магазин», которые присутствуют в последовательности проданных товаров (ранее мы приводили соответствующий фрагмент, использующий вызовы методов LINQ):

```
var r =
  from e1 in E
  join e2 in B
  on e1.art equals e2.art
  join e3 in D
  on e1.art + e1.shop equals e3.art + e3.shop
  join e4 in C
  on e1.code + e1.shop equals e4.code + e4.shop
  into ee4
  select new
  {
    cat_shop = e2.cat + " " + e1.shop,
    discount = (from e in ee4 select e.discount)
      .FirstOrDefault() * e3.price / 100
  }
    into e
    orderby e.cat_shop
    group e.discount by e.cat_shop
      into e
      select e.Key + " " + e.Sum();
```