МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Курский государственный университет»

кафедра программного обеспечения и администрирования

информационных систем

Отчёт

по контрольной работе № 2

# «Запросы LINQ к локальным спискам»

по дисциплине

**«**Объектно-ориентированные языки и системы»

|  |  |
| --- | --- |
| Выполнил: | студент группы 313  Мусонда Салиму |
| Проверил: | к.т.н., доцент  кафедры ПОиАИС  Макаров К.С. |

Курск

2020

# Цель: разработка в среде Microsoft Visual Studio консольного приложения для работы с локальными обобщёнными списками.

**Задачи**:

1. Изучить основные понятия лекции на тему «LINQ».
2. Выполнить задания (см. раздел Задания) в соответствии с вариантом.

**Теоретические сведения**

Одним из важнейших достижений в технологии разработки современных приложений является реализация запросов к базам данных с целью получения информации, подготовленной для восприятия пользователем. Помимо извлечения зачастую информацию требуется фильтровать, группировать и сортировать информацию, требуется также выполнять вычисления и другие операции. Все эти действия можно выполнять с помощью запросов LINQ.

LINQ (Language INtegrated Query – Язык интегрированных запросов) – это встроенный в среду Visual Studio язык, позволяющий извлекать и обрабатывать данные из разнообразных источников, информация в которых может иметь различные форматы данных.

Источниками данных могут быть как локальные коллекции, так и удаленные базы данных. Для извлечения данных могут применяться достаточно сложные условия поиска. Одновременно с реализацией запроса можно выполнять проекцию, фильтрацию, группировку и сортировку извлекаемых данных. Можно также реализовывать агрегирующие арифметические операции с коллекциями: нахождение максимальных и минимальных значений, подсчет сумм и средних арифметических значений и т. д.

При реализации запроса, возвращающего список, возможно появление новых типов (классов) данных. В частности, при объединении данных новая таблица содержит столбцы, извлеченные (частично или полностью) из исходных таблиц, а это означает появление объекта нового класса.

Исходными данными для операции LINQ могут быть один или два списка любого типа. Результат выполнения также может быть списком, но может быть и одиночным значением.

Информация для заполнения объектов классов хранится в файлах. Файлы создают в Приложении «Блокнот». Все данные для одного объекта записывают в одной строке.

В РФ дробная часть отделяется от целой с помощью запятой, поэтому для разграничения элементов в файлах ее применять нельзя (если предполагается применение дробных чисел). Можно использовать пробел (если в данном проекте строковые данные пробелов не имеют), точку с запятой и т. д. Ниже в примере запятая отделяет дробную часть числа от целой, а элементы разграничены точкой с запятой. Строковые данные в кавычки не заключают.

Пусть имеются таблицы следующей структуры.

|  |  |
| --- | --- |
| **Магазин** | **Товар** |
| Ключ | Ключ |
| Название | Название |
| Адрес | Цена |
|  | Количество |

Объявление классов для этих таблиц показано ниже.

class Vegetable

{

public int vKey { get; set; } //ключ

public string Name { get; set; } //название

public double Price { get; set; } //цена

public int Quan { get; set; } //количество

public Vegetable() {}

public Vegetable(int q, string n, double p) //конструктор

{

Name = n;

Price = p;

Quan = q;

}

}

Класс «Магазин»

class Shop

{

public string Name { get; set; } //название товара, вместо ключа

public string ShopName { get; set; } //название магазина

public string Street { get; set; }

public Shop() { } //пустой конструктор

public Shop(string s, string c, string n) //конструктор с параметрами

{

ShopName = s;

Street = c;

Name = n;

}

public Shop(Shop it) //конструктор, принимающий объект

{

ShopName = it.ShopName;

Street = it.Street;

Name = it.Name;

}

}

Для каждой таблицы создается автономный файл, и ввод данных для нее выполняется отдельно.

Если требуется, чтобы одна таблица содержала вложенную или объединенную таблицу, то для создания такого списка в оба класса, представляющих таблицы, добавляются целочисленные ключи, по которым выполняется объединение классов с помощью запросов LINQ Join или GroupJoin. В некоторых случаях (когда содержимое поля может быть ключом) добавление в класс специального ключа необязательно. Преимущество этого способа – возможность группировки по элементам вложенного списка, поскольку посредством объединения списков можно главным списком сделать вложенный.

Ниже приведен пример оформления файлов (в начале каждой строки целочисленный ключ):

Файл 1 (магазины)

1; Ашан; ул. Широкая;

2; Дикси; ул. Полярная;

Файл 2 (товары)

1; Помидор; 83,5; 1

1; Огурец; 65; 2

2; Морковь; 35,2; 3

2; Свекла; 14,5; 2

2; Лук; 21,4; 3

(В магазине «Ашан» 2 товара, в магазине «Дикси» 3).

Ниже приведен метод для формирования обобщенного списка для одного класса. (В данном примере имя класса Vegetable (Овощи). В заданиях на контрольную работу должно быть 3 или 4 класса, соответственно измените имена классов, пути к файлам, объявления списков и количество полей в классах). Метод оформлен для формирования одного списка и вызывается несколько раз в цикле, передавая в него путь к файлу (path), индекс списка (n) и количество элементов в строке файла (m). Списки объявляют глобально (в классе Program). В метод передается индекс массива, содержащий количество элементов в строке файла (m). Раздельная обработка каждого списка (по индексу) обусловлена тем, что объект каждого класса отличается количеством элементов, их типами и порядком следования.

class Program

{

static List<Shop> lst1 = new List<Shop>();

static List<Vegetable> lst2 = new List<Vegetable>();

static string[] path = { "D:\\List1.txt", "D:\\List2.txt" };

static void Main(string[] args)

{

File\_Load(path); //считывание файла

//обработка в соответствии с заданием контрольной работы

Console.ReadKey();

}

static void File\_Load(string[] path, int n, int m)

{

if(n == 0) //первая таблица

{

StreamReader FileIn = new StreamReader(path[0], Encoding.Default);

string str;

while((str = FileIn.ReadLine()) != null) //пока не конец файла

{

string[] ms = new string[m]; //m - количество полей в классе

ms = str.Split(';'); //расщепление на массив строк

//добавление строки через конструктор с параметрами

lst1.Add(new Shop(ms[0], ms[1], ms[2]));

}

}

//при повторном вызове метода n = 1, вторая таблица, операторы if и while опущены

if(n == 1)

{

StreamReader FileIn = new StreamReader(path[1], Encoding.Default);

string str;

while ((str = FileIn.ReadLine()) != null) //пока не конец файла

{

string[] ms = new string[m]; //m - количество полей в этом классе

ms = str.Split(';'); //расщепление на массив строк

//добавление строки через конструктор с параметрами

lst2.Add(new Vegetable(Convert.ToInt32(ms[0]), ms[1], Convert.ToDouble(ms[2]))); //второй список

}

}

}

}

Для конкретного задания требуется описать классы и соответственно изменить поля и операторы добавления элементов.

Запросы в каждом задании постоянные. Для их обработки оформляется оператор switch. Реализация запроса оформляется в виде группы case, вызывающей метод для обработки запроса.

Для моделирования работы с базой данных желательно запросы задавать номером с описанием запроса. После выбора номера нажимают на клавишу Enter. Желающие могут попробовать изменить программу так, чтобы нажатия на клавишу Enter после выбора номера не требовалось. (с использованием обработки события keybd\_event).

Для вызова запросов по номеру рекомендуется реализовать приведенную ниже программу. Перед запуском цикла обработки запросов в цикле формируются обобщенные списки. С этой целью объявляются списки (их количество зависит от задания, в примере 2), и созданы массивы, содержащие пути к трем файлам и количество полей в каждом классе. Именам списков желательно задавать осмысленные имена. class Program

{

static List<Shop> lst1 = new List<Shop>();

static List<Vegetable> lst2 = new List<Vegetable>();

static string[] path = { "D:\\List1.txt", "D:\\List2.txt" };

static int[] m = { 3, 6, 5 }; //количество полей в классе для каждого класса

static void Main(string[] args)

{

// File\_Load(path); //считывание файла

for (int i=0; i<2; ++i)

{

File\_Load(path[i], i, m[i]); //p[i] - пути, i - индекс списка, m[i] - количество полей в классе

}

int n = -1; //позиция в switch

while(true)

{

Frame(); //вывод списка для выбора запоса

if(!int.TryParse(Console.ReadLine(),out n))

{

Console.WriteLine("Это не цифра, нажмите на Enter");

Console.ReadLine();

Console.Clear();

continue;

}

switch(n)

{

case 1:

Console.Clear();

//вызов метода для реализации запроса для позиции 1

Console.WriteLine("Запрос окончен, нажмите любую клавишу");

Console.ReadKey();

Console.Clear();

continue;

case 2:

Console.Clear();

//вызов метода для реализации запроса для позиции 2

Console.WriteLine("Запрос окончен, нажмите любую клавишу");

Console.ReadKey();

Console.Clear();

continue;

//обработка других запросов

case 9:

Console.WriteLine("Выход из программы - да, нет - любая клавиша");

string st = Console.ReadLine();

if (st == "да")

{

Environment.Exit(0);

}

else

{

Console.Clear();

continue;

}

continue;

default:

Console.Clear();

Console.WriteLine("Введённый № отсутствует, нажмите любую клавишу");

Console.ReadKey();

Console.Clear();

continue;

}

}

//обработка в соответствии с заданием контрольной работы

Console.ReadKey();

}

static void Frame() //метод для выбора списка запросов

{

Console.WriteLine("\tВыберете номер задачи\n");

Console.WriteLine("1 - Получить справку о конкретном клиенте\n");

Console.WriteLine("2 - Суммарная стоимость товара в отделе\n");

Console.WriteLine("\tДругие запросы");

Console.WriteLine("\n9 - Выход из программы");

}

static void File\_Load(string path, int n, int m)

{

if(n == 0) //первая таблица

{

StreamReader FileIn = new StreamReader(path, Encoding.Default);

string str;

while((str = FileIn.ReadLine()) != null) //пока не конец файла

{

string[] ms = new string[m]; //m - количество полей в классе

ms = str.Split(';'); //расщепление на массив строк

//добавление строки через конструктор с параметрами

lst1.Add(new Shop(ms[0], ms[1], ms[2]));

}

}

//при повторном вызове метода n = 1, вторая таблица, операторы if и while опущены

if(n == 1)

{

StreamReader FileIn = new StreamReader(path, Encoding.Default);

string str;

while ((str = FileIn.ReadLine()) != null) //пока не конец файла

{

string[] ms = new string[m]; //m - количество полей в этом классе

ms = str.Split(';'); //расщепление на массив строк

//добавление строки через конструктор с параметрами

lst2.Add(new Vegetable(Convert.ToInt32(ms[0]), ms[1], Convert.ToDouble(ms[2]))); //второй список

}

}

}

}

Инициализация объектов и списков

Инициализация объекта класса может выполняться либо с помощью свойств ({ get; set; }), либо с помощью конструктора. Если инициализация выполняется с помощью конструктора, описания свойств не требуются. Если для инициализации применяются свойства, конструктор не нужен. В некоторых случаях система требует описание пустого конструктора, который приведен в описаниях классов.

Если класс содержит массив, то инициализация такого массива выполняется только с помощью конструктора. Ниже приведен пример объявления класса, содержащего массив, и конструктор для инициализации этого класса.

class AZS

{

public string Firm;

public string[] Benz = new string[3];

public AZS(string f, params string[] b)

{

Firm = f;

Benz[0] = b[0];

Benz[1] = b[1];

Benz[2] = b[2];

}

}

Обратите внимание на модификатор params. Такой модификатор может быть только один, поэтому массив в классе, как правило, также один. Но если два массива имеют одинаковый тип, то индексы переменной с модификатором params (в примере g) можно продолжить для инициализации второго массива (в котором индексы от 0, а индексы g следуют за максимальным индексом первого массива).

Инициализация списка с элементами, содержащими массив, приведена ниже.

static List<AZS> az = new List<AZS>()

{

new AZS("Роснефть", "98", "92"), //первый элемент списка

new AZS("ТНК", "98", "95"), //второй элемент списка

};

Примеры запросов

Сохранять результаты запросов не требуется. После реализации каждого запроса его данные выводятся на экран. После просмотра выполняется стирание экрана, и информация будет восстановлена при последующем запросе.

Ниже для справки приведены наиболее распространенные запросы LINQ.

Сортировка, проекция, фильтрация

В примерах запросов: lst обобщенный список для класса Vegetable (см. выше). w – имя переменной для лямбда-выражения (это имя может быть любым другим).

Выборка всех столбцов:

var qr = lst1.Select(w => w);

Выборка всех столбцов с сортировкой (по возрастанию):

var qr = lst1.Select(w => w).OrderBy(w => w.Street);

Выборка двух столбцов (проекция):

var qr = lst1.Select(w => new { w.Name, w.ShopName });

При выполнении запроса GroupJoin обычно получается вложенный список. Запрос Select выполняет выборку только на одном уровне. Поэтому при запросе Select вложенный список виден, но доступ к его полям отсутствует. Чтобы получить доступ к полям вложенного списка внутри анонимного метода применяют запрос SelectMany:

var qr = ls.Select(rl => new {k = rl.Key, slst = rl.SelectMany(c => c.lst) });

Здесь ls содержит вложенный список, k – новое имя для ключа группировки, slst – новое имя вложенного списка; теперь его поля доступны в операторе foreach.

Фильтрация (выборка строк по условию)

В одном запросе можно задать условие только для одного свойства (поля).

a) Попадание в диапазон значений:

var qr2 = lst2.Where(w => w.Quan >= 2 && w.Quan < 5);

b) Добавить условие для второго свойства (Сравнение для второго свойства записывается через точку после первого сравнения. Это обеспечивает операцию И (&&) между логическими операциями для каждого поля).

var qr2 = lst2.Where(w => w.Quan >= 2 && w.Quan < 5).Where(b => b.Price < 30);

Если во второй проверке b заменить на a, ничего не изменится.

Выделение части списка и поиск

Операторы Take, Skip, TakeWhile и SkipWhile позволяют либо выбирать строки из начала списка, либо пропускать.

a) Взять первые три записи от начала списка:

var qr3 = lst2.Take(3);

b) Выбрать с начала списка по условию:

var qr3 = lst2.TakeWhile(w => w.Price > 50);

c) Поиск одиночного значения

var qr3 = lst2.FirstOrDefault(w => w.Name == " Помидор");

Vegetable vg\_ = (Vegetable)qr3;

Console.WriteLine(vg\_.Name + " по цене: " + vg\_.Price + " р.; " + "\n");

Запрос FirstOrDefault выбирает первый элемент в списке с заданным для поиска значением. Если такой элемент не найден, то выводится первый элемент из списка. Обратите внимание, при поиске выбирается объект класса, а не одиночное значение.

Запрос LastOrDefault выбирает последний элемент в списке с заданным для поиска значением. Если такой элемент не найден, то выводится последний элемент из списка.

Для поиска всех элементов с заданным для поиска значением можно применять запрос FindAll, но, наверное, лучше в этом случае применять запрос Where.

Операции с коллекциями

При работе с коллекциями выполняются операции, характерные для операций с множествами, а также операции по группировке и объединению коллекций. Операции могут относиться к коллекциям с одинаковым типом данных и к коллекциям с разными типами данных. К первой группе относятся перечисленные ниже операции:

Concat Объединение (конкатенация) двух коллекций

Distinct Исключение из списка повторяющихся элементов

Except Удаление из одного списка элементов другого списка

Intersect Пересечение двух коллекций

Union Объединение двух коллекций с удалением повторяющихся

элементов

Операции с множествами

Для выполнения операций с коллекциями для типов данных пользователя требуется сравнивать между собой объекты класса. Поэтому класс следует изменить (добавить класс интерфейса и добавить методы для сравнения свойств). Класс приобретет вид:

public bool Equals(Vegetable other)

{

if(Object.ReferenceEquals(other, null)) //проверить, не имеет ли сравниваемый объект значение null

{

return false;

}

if(Object.ReferenceEquals(this, other)) //проверить, не ссылаются ли сравниваемые объекты на одни и те же данные

{

return true;

}

return Name.Equals(other.Name) && Price.Equals(other.Price); //проверить, равны ли свойства товаров

}

// override object.GetHashCode

public override int GetHashCode()

{

int hashVegName = Name == null ? 0 : Name.GetHashCode(); //получить хеш-код для поля Name, если оно не равно null

int hashVegPrice = Price.GetHashCode(); //получить хеш-код для поля Price

return hashVegName ^ hashVegPrice; //вычислить хеш-код для товара

}

Теперь запрос, например, на удаление повторяющихся элементов имеет вид:

var qr4 = lst2.Distinct();

В данном примере удаляются элементы, у которых совпадает название и цена.

Объединение и группировка коллекций

При объединении коллекций с данными пользователя возможно как внутреннее, так и внешнее объединение. Для объединения таблиц должен быть выбран ключ. При внутреннем объединении каждой строке в одной таблице строго соответствует одна строка в другой таблице. Внешнее левое объединение позволяет собрать новую таблицу из столбцов так, что в новой таблице повторяются строки из одной таблицы с одинаковым значением ключа в другой таблице. Если после объединения выполнить группировку (по повторяющимся строкам), получается аналог отношения «один ко многим». Ниже приведен пример левого внешнего объединения (рисунок 1).

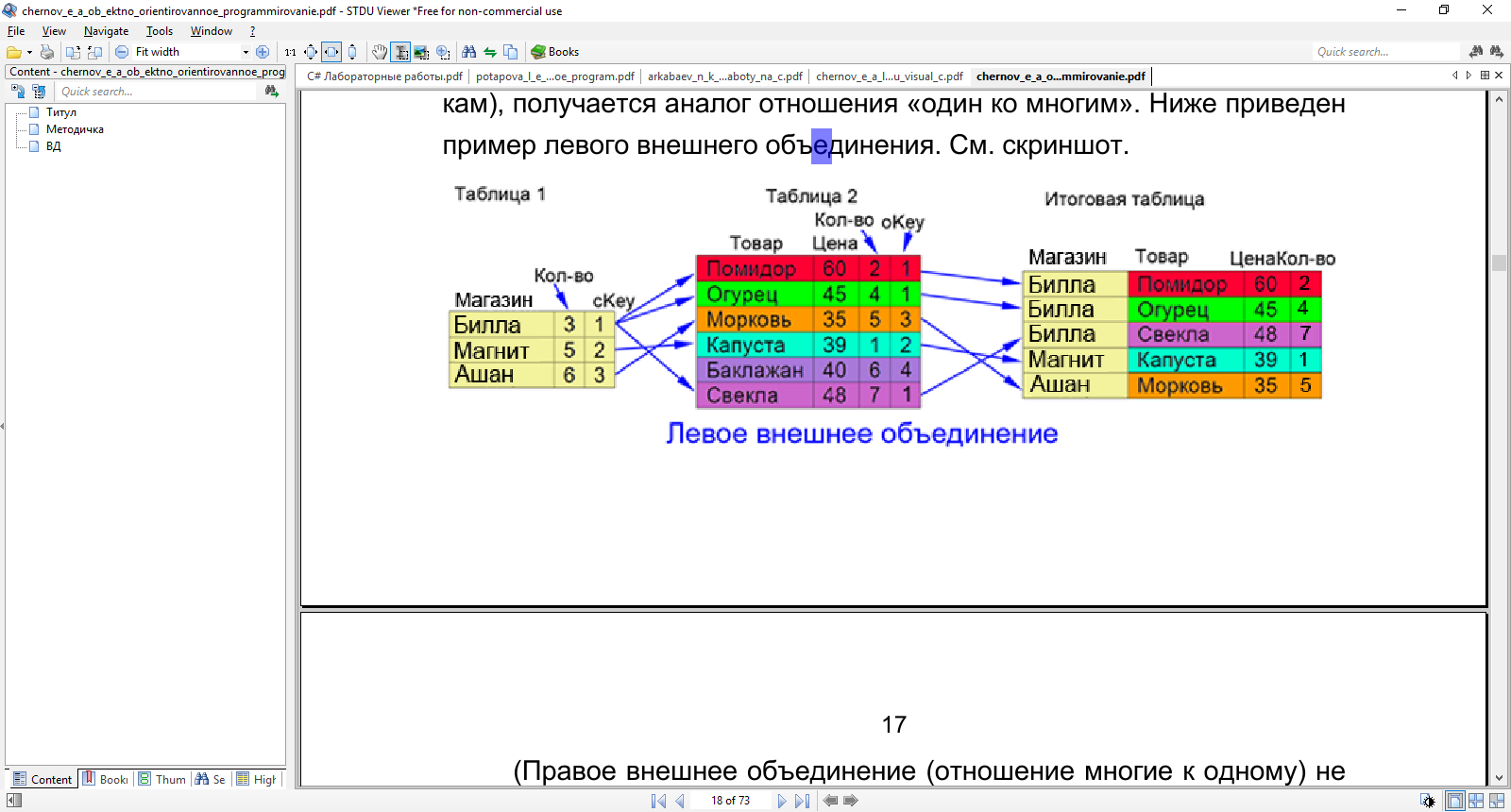


Рисунок 1 – Левое внешнее объединение

(Правое внешнее объединение (отношение многие к одному) не рассматривается).

Ниже приведен пример объединения двух классов (Shop и Vegetable) по ключам cKey (магазин), oKey (товар). (Shop – левый список, переменная w для левого списка, переменная b для правого списка)

var qr5 = lst1.Join(lst2, w => w.cKey, b => b.vKey, (w,b) => new

{

w.ShopName,

b.Name,

b.Price,

b.Quan,

});

string[] str = { "Магазин", "Овощ", "Цена", "Количество" }; //заголовки

Console.WriteLine("{0}\t{1}\t\t{2}\t{3}\n", str[0], str[1], str[2], str[3]);

foreach(var v in qr5) //вывод новой таблицы

{

Console.WriteLine("{0}\t{1,-10}\t{2}\t{3}", v.ShopName, v.Name, v.Price, v.Quan);

}

При выводе таблицы обратите внимание на переменную v, она одинакова для свойств из левой и правой таблиц, поскольку вся таблица теперь единая (объединенная).

Принципиальным является получение нового типа данных (новой таблицы), запросы LINQ к которой позволяют реализовать достаточно сложные операции, в которых участвуют две таблицы.

Ниже приведен пример применения группировки к итоговой таблице (в примере операции Join выше была получена таблица с именем qr).

Обратите внимание: после группировки из результата извлекается ключ и список для каждого ключа.

var grp = qr.GroupBy( w => w.ShopName).Select(rs =>

new

{

kgr = rs.Key, // Ключ группы

sng = rs // Список группы

});

foreach (var res in grp)

{

Console.WriteLine("\nЗаказчик: " + res.kgr); // Вывод ключей

foreach (var sp in res.sng) // Цикл вывода списка группы

Console.WriteLine("{0}\t{1}\t{2} ", sp.Name, sp.Price, sp.Quan);

}

Имена переменных res и sp в операторах foreach произвольные, но переменная res из наружного цикла позволяет с помощью точечной нотации обратиться к списку, относящемуся к группе.

К списку группы с помощью точечной нотации можно применять агрегирующие запросы.

Группировка может выполняться по разным ключам и может иметь несколько уровней.

Агрегирующие запросы

Основные операции для агрегирующих запросов: Average (Среднее значение), Aggregate (операции одного элемента с последовательным по значению другим), Count (количество элементов в списке), Max, Min, Sum. Эти операции возвращают одиночное значение для конкретного списка данных. Например, оператор:

int n = qs.Count();

подсчитывает количество элементов в списке qs.

**Экспериментальные результаты**

Задания

Требования к программе:

– информация для списков должна сохраняться в текстовых файлах;

– каждый из списков должен содержать не менее 10 инициализированных объектов класса.

Структура сущностей приведена в описании каждого индивидуального задания. Реализация классов может предусматривать наличие конструктора с параметрами (в зависимости от вида запросов), но может быть выполнена с помощью свойств в форме {get; set;}. Свойств должно быть не меньше, чем указано в задании.

Разрабатываемое приложение должно обрабатывать не менее шести запросов. Запросы выполняются в форме LINQ to Objects. Тексты запросов на русском языке должны представляться на экране в виде списка, и каждый из запросов может быть выбран посредством выбора номера его позиции в списке. После выполнения запроса на экран снова выводится список для выбора другого запроса.

Одна из позиций в списке должна обеспечивать штатный выход из программы.

Текст программы

using System;

public class player

{

public int key;

public string FIO;

public int age;

public string role;

public int number;

public player(int a,string b,int c,string d,int e)

{

key = a;

FIO = b;

age = c;

role = d;

number = e;

}

public void Show()

{

Console.WriteLine("ключ:" + key + "\n");

Console.WriteLine("ФИО:" + FIO + "\n");

Console.WriteLine("возраст:" + age + "\n");

Console.WriteLine("амплуа:" + role + "\n");

Console.WriteLine("номер:" + role + "\n");

Console.WriteLine("\n");

}

}

using System;

public class stadium

{

public int key;

public string Name;

public string City;

public int Capacity;

public stadium(int a,string b,string c,int d)

{

key = a;

Name = b;

City = c;

Capacity = d;

}

public void Show()

{

Console.WriteLine("ключ:" + key + "\n");

Console.WriteLine("Название:" + Name + "\n");

Console.WriteLine("город:" + City+ "\n");

Console.WriteLine("вместимость:" + Capacity + "\n");

Console.WriteLine("\n");

}

}

using System;

public class command

{

public string Name;

public string city;

public string Trainer;

public int raiting;

public int[] players;

public int[] stadiums;

public command(string a, string b, string c, int d, int n, int n1, int[] e, int[] e1)

{

Name = a;

city = b;

Trainer = c;

raiting = d;

players = new int[n];

for (int i = 0; i < n; i++)

players[i] = e[i];

stadiums = new int[n1];

for (int i = 0; i < n1; i++)

stadiums[i] = e1[i];

}

public void Show()

{

Console.WriteLine("Название команды:"+Name+"\n");

Console.WriteLine("город:" + city + "\n");

Console.WriteLine("тренер:" + Trainer + "\n");

Console.WriteLine("рейтинг:" + raiting + "\n");

Console.Write("игроки: ");

for(int i=0;i<players.Length;i++)

Console.Write(players[i]+" ");

Console.Write("\n");

Console.Write("стадионы: ");

for (int i = 0; i < stadiums.Length; i++)

Console.Write(stadiums[i]+" ");

Console.Write("\n");

}

}

using System;

public class match

{

public int key;

public string command1;

public string command2;

public string date;

public int count1;

public int count2;

public double price;

public int stadium;

public match(int a,string b,string c,string d,int e,int e1,int f,double g)

{

key = a;

command1 =b;

command2=c;

date = d;

count1 = e;

count2 = e1;

stadium = f;

price = g;

}

public void Show()

{

Console.WriteLine("Ключ: " + key + "\n");

Console.WriteLine("Команды: " + command1 +" и "+ command2 +"\n");

Console.WriteLine("дата: " + date + "\n");

Console.WriteLine("счёт: " + count1 + "-" + count2 + "\n");

Console.WriteLine("цена: " + price + "\n");

Console.WriteLine("стадион: " + stadium + "\n");

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

namespace ConsoleApp2

{

class Program

{

class v

{

public string c;

public int c1;

public v(string a,int b)

{

c = a;

c1 = b;

}

public void add(int x)

{

this.c1 += x;

}

public void sub(int x)

{

this.c1 -= x;

}

public void Show()

{

Console.WriteLine(c);

Console.WriteLine(c1);

}

}

public static void keys()

{

Console.WriteLine("Для выхода из программы нажмите 0");

Console.WriteLine("Для вывода списков нажмите 1");

Console.WriteLine("Для вывода дат встреч команды, противников и счёта нажмите 2");

Console.WriteLine("Для вывода списка игроков, участвовавших во встрече, нажмите 3");

Console.WriteLine("Для вывода цены билета на матч нажмите 4");

Console.WriteLine("Для вывода команд с наилучшей и наихудшей разницей забитых мячей нажмите 5");

Console.WriteLine("Для вывода команд,занявших призовые места, нажмите 6");

Console.WriteLine("Для вывода расписания игр по стадионам нажмите 7");

Console.WriteLine("Для добавления игрока нажмите 8");

Console.WriteLine("Для удаления игрока нажмите 9");

Console.WriteLine("Для добавления матча нажмите a");

Console.WriteLine("Для удаления матча нажмите b");

}

static void Main(string[] args)

{

string str="";

string[] a;

List<command> list1 = new List<command>();

List<player> list2 = new List<player>();

List<stadium> list3 = new List<stadium>();

List<match> list4 = new List<match>();

List<v> list5 = new List<v>();

StreamReader f=new StreamReader("E:\\С#\\files\\стадионы.txt", System.Text.Encoding.Default);

while ((str=f.ReadLine())!=null)

{

a = str.Split(";");

list3.Add(new stadium(Convert.ToInt32(a[0]), a[1], a[2], Convert.ToInt32(a[3])));

}

f.Close();

f= new StreamReader("E:\\С#\\files\\игроки.txt", System.Text.Encoding.Default);

while ((str = f.ReadLine()) != null)

{

a = str.Split(";");

list2.Add(new player(Convert.ToInt32(a[0]), a[1], Convert.ToInt32(a[2]), a[3], Convert.ToInt32(a[4])));

}

f.Close();

f = new StreamReader("E:\\С#\\files\\команды.txt", System.Text.Encoding.Default);

string[] b;

string[] c;

int[] b1;

int[] c1;

while ((str = f.ReadLine()) != null)

{

a = str.Split(";");

b = a[4].Split(":");

c = a[5].Split("!");

b1 = new int[b.Length];

c1 = new int[c.Length];

for (int i = 0; i < c1.Length; i++)

c1[i] = Convert.ToInt32(c[i]);

for (int i = 0; i < b.Length; i++)

b1[i] = Convert.ToInt32(b[i]);

list1.Add(new command(a[0], a[1], a[2],Convert.ToInt32(a[3]),b1.Length,c1.Length, b1, c1));

}

f.Close();

f = new StreamReader("E:\\С#\\files\\матчи.txt", System.Text.Encoding.Default);

string []b2;

while ((str = f.ReadLine()) != null)

{

b2= str.Split(";");

var j1 = from p in list1 where p.Name == b2[1] select p;

command k1 = j1.First();

var j2 = from p in list1 where p.Name == b2[2] select p;

var r = from p in list1 where true select p.raiting;

command k2 = j2.First();

double price = 0;

int minr = r.Max();

double r1, r2;

r1 = k1.raiting;

r2 = k2.raiting;

double t = 0;

if (r2 < r1)

{

t = r1;

r1 = r2;

r2 = t;

}

if (r1 <= 3 && r2 <= 3)

price = 150;

else if (r1 >= 3 && r1 < minr - 2 && r2 >= 3 && r2 < minr - 2)

price = 120;

else if (r1 >= minr - 2 && r2 >= minr - 2)

price = 90;

else if (r1 <= 3 && r2 >= 3 && r2 < minr - 2)

price = (150 + 120) / 2;

else if (r1 <= 3 && r2 > minr - 2)

price = (150 + 90) / 2;

else if (r1 >= 3 && r1 < minr - 2 && r2 >= minr - 2)

price = (120 + 90) / 2;

var s1 = from p in list3 where p.key == Convert.ToInt32(b2[6]) select p;

price -= s1.First().Capacity;

list4.Add(new match(Convert.ToInt32(b2[0]), b2[1], b2[2], b2[3], Convert.ToInt32(b2[4]), Convert.ToInt32(b2[5]),Convert.ToInt32(b2[6]),price));

}

f.Close();

ConsoleKeyInfo n = new ConsoleKeyInfo();

StreamWriter pf;

while (true)

{

keys();

n =Console.ReadKey();

char s = n.KeyChar;

switch (s)

{

case '0':

return;

case '1':

{

Console.WriteLine("\n");

Console.WriteLine("Команды");

foreach (command l in list1)

l.Show();

Console.WriteLine("\n");

Console.WriteLine("Игроки");

foreach (player l in list2)

l.Show();

Console.WriteLine("\n");

Console.WriteLine("Стадионы");

foreach (stadium l in list3)

l.Show();

Console.WriteLine("\n");

Console.WriteLine("Матчи");

foreach (match l in list4)

l.Show();

break;

}

case '2':

{

string command1;

Console.WriteLine("\nВведите команду");

command1 = Console.ReadLine();

var d = from p in list4 where p.command1 == command1 || p.command2 == command1 select p;

if(!d.Any())

{

Console.WriteLine("Матчи с участием данной команды отсутствуют");

break;

}

foreach (match v in d)

{

string d1;

if (v.command1 == command1)

d1 = v.command2;

else d1 = v.command1;

Console.WriteLine("Противник: " + d1);

Console.WriteLine("Дата: " + v.date);

Console.WriteLine("Счёт: " + v.count1 + "-" + v.count2);

}

break;

}

case '3':

{

string date1;

int stadium2;

Console.WriteLine("\nВведите дату");

date1 = Console.ReadLine();

Console.WriteLine("Введите стадион");

stadium2 = Convert.ToInt32(Console.ReadLine());

var d2 = from p in list4 where p.date == date1 && p.stadium == stadium2 select p;

if (!d2.Any())

{

Console.WriteLine("Матчи с данной датой на данном стадионе отсутствуют");

break;

}

var h = from p in d2 from p1 in list1 where p1.Name == p.command1 || p1.Name == p.command2 select p1.players;

//var h1=from p in h

foreach (int[] v in h)

{

foreach(int v1 in v)

{

var h1 = from p in list2 where p.key == v1 select p;

foreach (player v2 in h1)

v2.Show();

//int x = 3;

//x = x;

}

}

break;

}

case '4':

{

string command1;

string command2;

Console.WriteLine("\nВведите первую команду");

command1 = Console.ReadLine();

Console.WriteLine("Введите вторую команду");

command2 = Console.ReadLine();

var v = from p in list4 where (p.command1 == command1 && p.command2 == command2) || (p.command2 == command1 && p.command1 == command2) select p;

if (!v.Any())

{

Console.WriteLine("Матчи между данными командами отсутствуют");

break;

}

Console.WriteLine(v.First().price);

break;

}

case '5':

{

list5.Clear();

Console.WriteLine("\n");

var y = from p in list4 where true select p.command1;

var y1 = from p in list4 where true select p.command2;

y = y.Distinct();

y1 = y1.Distinct();

foreach (string n1 in y)

{

var y2 = from p in list4 where p.command1 == n1 select p.count1;

int z = y2.Sum();

list5.Add(new v(n1, z));

}

foreach (string n2 in y1)

{

var y2 = from p in list4 where p.command2 == n2 select p.count2;

int z = y2.Sum();

int t = -1;

for (int i = 0; i < list5.Count; i++)

{

if (list5[i].c == n2)

t = i;

}

if (t >= 0)

list5[t].add(z);

}

foreach (string n3 in y)

{

var y2 = from p in list4 where p.command1 == n3 select p.count2;

int z = y2.Sum();

int t = -1;

for (int i = 0; i < list5.Count; i++)

{

if (list5[i].c == n3)

t = i;

}

if (t >= 0)

list5[t].sub(z);

}

foreach (string n4 in y1)

{

var y2 = from p in list4 where p.command2 == n4 select p.count1;

int z = y2.Sum();

int t = -1;

for (int i = 0; i < list5.Count; i++)

{

if (list5[i].c == n4)

t = i;

}

if (t >= 0)

list5[t].sub(z);

}

var x = from i in list5 orderby i.c1 select i;

int min = x.First().c1;

int max = x.First().c1;

foreach (v v1 in x)

{ if (v1.c1 < min)

min = v1.c1;

if (v1.c1 > max)

max = v1.c1;

}

Console.WriteLine("наихудшая разница");

foreach (v v1 in x)

{

if (v1.c1 == min)

v1.Show();

}

Console.WriteLine("наилучшая разница");

foreach (v v1 in x)

{

if (v1.c1 == max)

v1.Show();

}

break;

}

case '6':

{

Console.WriteLine("\n");

var g = from p in list1 where p.raiting < 4 select p;

foreach (command v in g)

{

v.Show();

}

break;

}

case '7':

{

int stadium1;

Console.WriteLine("\nВведите стадион");

stadium1 = Convert.ToInt32(Console.ReadLine());

var g1 = from p in list4 where p.stadium == stadium1 select p;

if (!g1.Any())

{

Console.WriteLine("Матчи,назначенные на данный стадион, отсутствуют");

break;

}

foreach (match v in g1)

{

v.Show();

}

break;

}

case '8':

{

StreamReader fr = new StreamReader("E:\\С#\\files\\игроки.txt");

string s1 = fr.ReadLine();

fr.Close();

pf = new StreamWriter("E:\\С#\\files\\игроки.txt", true, System.Text.Encoding.Default);

if (s1 != null)

pf.WriteLine();

//var y1 = from p in list2 where true select p;

Console.WriteLine("\nВведите ключ:");

int pk = Convert.ToInt32(Console.ReadLine());

bool z = false;

foreach(player v in list2)

{

if (pk == v.key)

z = true;

}

if(z==true)

{

Console.WriteLine("Ключи не должны совпадать");

break;

}

pf.Write(pk + ";");

Console.WriteLine("Введите ФИО:");

string pfio = Console.ReadLine();

pf.Write(pfio + ";");

Console.WriteLine("Введите возраст:");

int pa = Convert.ToInt32(Console.ReadLine());

pf.Write(pa + ";");

Console.WriteLine("Введите амплуа:");

string pr = Console.ReadLine();

pf.Write(pr + ";");

Console.WriteLine("Введите номер:");

int pn = Convert.ToInt32(Console.ReadLine());

pf.Write(pn);

//pf.WriteLine("aaa");

pf.Close();

break;

}

case '9':

{

List<player> list6 = new List<player>();

Console.WriteLine("\nВведите ключ:");

int pk = Convert.ToInt32(Console.ReadLine());

player p = list2.Find(p => p.key == pk);

if (p == null)

{

Console.WriteLine("Записи о данном игроке отсутствуют");

break;

}

list2.Remove(p);

//foreach (player r in list2)

//r.Show();

pf = new StreamWriter("E:\\С#\\files\\игроки.txt", false, System.Text.Encoding.Default);

foreach (player v in list2)

{

pf.Write(v.key + ";");

pf.Write(v.FIO + ";");

pf.Write(v.age + ";");

pf.Write(v.role + ";");

if (v != list2.Last())

pf.WriteLine(v.number);

else pf.Write(v.number);

}

pf.Close();

// if (r.key != p.key)

// l

break;

}

case 'a':

{

StreamReader fr = new StreamReader("E:\\С#\\files\\матчи.txt");

string s1 = fr.ReadLine();

fr.Close();

pf = new StreamWriter("E:\\С#\\files\\матчи.txt", true, System.Text.Encoding.Default);

if (s1 != null)

pf.WriteLine();

Console.WriteLine("\nВведите ключ:");

int k = Convert.ToInt32(Console.ReadLine());

bool z = false;

foreach (match v in list4)

{

if (k == v.key)

z = true;

}

if (z == true)

{

Console.WriteLine("Ключи не должны совпадать");

break;

}

z = false;

Console.WriteLine("Введите первую команду");

string command1 = Console.ReadLine();

var y2 = from p in list1 where true select p;

z = false;

foreach (command v in y2)

{

if (v.Name == command1)

z = true;

}

if (z == false)

{

Console.WriteLine("Данные об этой команде отсутствуют");

break;

}

z = false;

Console.WriteLine("Введите вторую команду");

string command2 = Console.ReadLine();

foreach (command v in y2)

{

if (v.Name == command2)

z = true;

}

if (z==false)

{

Console.WriteLine("Данные об этой команде отсутствуют");

break;

}

z = false;

if(command1==command2)

{

Console.WriteLine("Матч не может проводиться между одинаковыми командами");

break;

}

Console.WriteLine("Введите дату");

string date = Console.ReadLine();

Console.WriteLine("введите счёт первой команды");

int count1= Convert.ToInt32(Console.ReadLine());

Console.WriteLine("введите счёт второй команды");

int count2 = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("введите стадион");

int stadium = Convert.ToInt32(Console.ReadLine());

var y3 = from p in list3 where true select p;

foreach (stadium v in y3)

{

if (v.key == stadium)

z = true;

}

if (z == false)

{

Console.WriteLine("Данные об этом стадионе отсутствуют");

break;

}

pf.Write(k + ";");

pf.Write(command1 + ";");

pf.Write(command2 + ";");

pf.Write(date + ";");

pf.Write(count1 + ";");

pf.Write(count2 + ";");

pf.Write(stadium + ";");

pf.Close();

break;

}

case 'b':

{

List<match> list6 = new List<match>();

Console.WriteLine("\nВведите ключ:");

int pk = Convert.ToInt32(Console.ReadLine());

match p = list4.Find(p => p.key == pk);

if (p == null)

{

Console.WriteLine("Записи о данном матче отсутствуют");

break;

}

list4.Remove(p);

//foreach (player r in list2)

//r.Show();

pf = new StreamWriter("E:\\С#\\files\\матчи.txt", false, System.Text.Encoding.Default);

foreach (match v in list4)

{

pf.Write(v.key + ";");

pf.Write(v.command1 + ";");

pf.Write(v.command2 + ";");

pf.Write(v.date + ";");

pf.Write(v.count1 + ";");

pf.Write(v.count2 + ";");

if (v != list4.Last())

pf.WriteLine(v.stadium);

else pf.Write(v.stadium);

}

pf.Close();

// if (r.key != p.key)

// l

break;

}

default:

Console.WriteLine("\nНеверная команда");

break;

}

}

}

}

}

Тестирование задачи 1 представлено на рисунках 1-11.

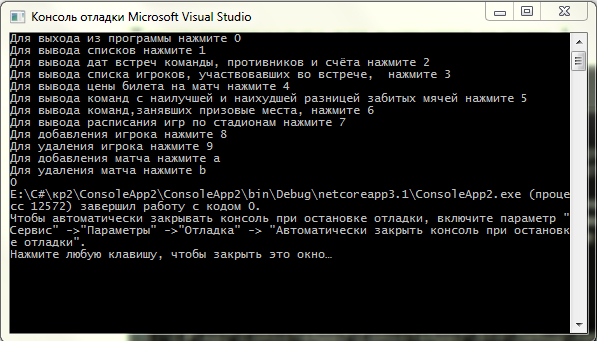


Рисунок 1 – Тест 1 задачи 1

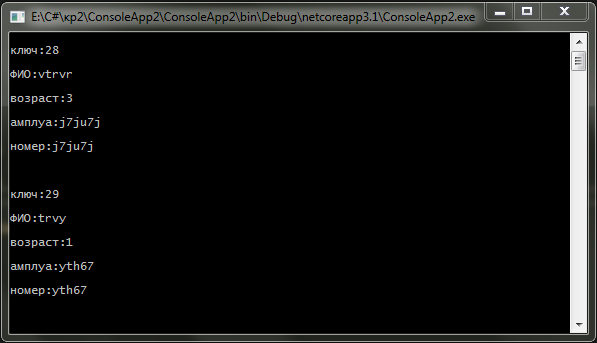


Рисунок 2 – Тест 2 задачи 1

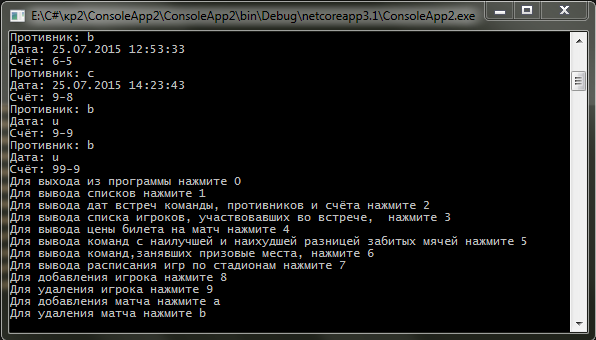


Рисунок 3 – Тест 3 задачи 1

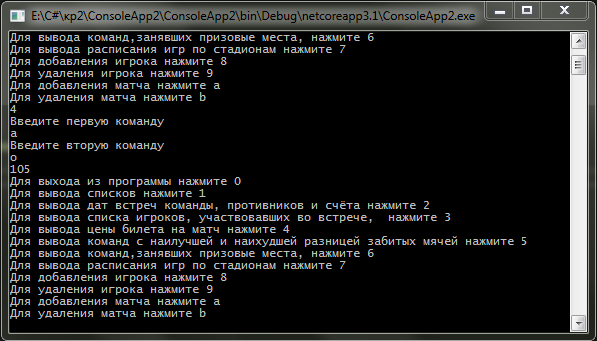


Рисунок 4 – Тест 4 задачи 1

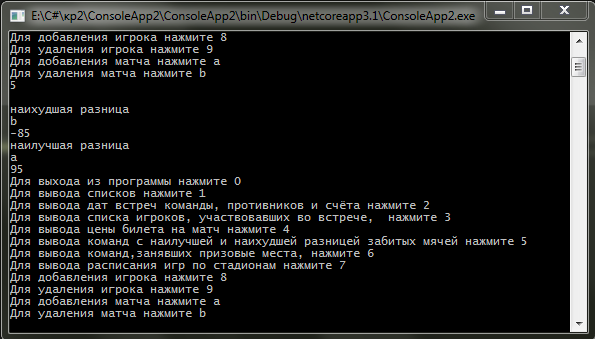


Рисунок 5 – Тест 5 задачи 1

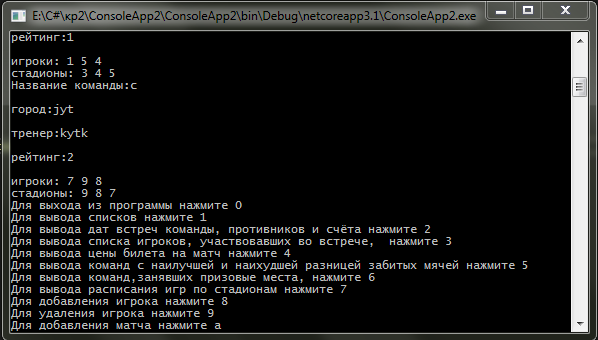


Рисунок 6 – Тест 6 задачи 1

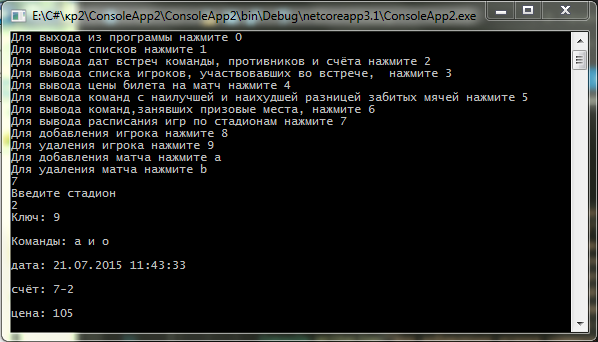


Рисунок 7 – Тест 7 задачи 1

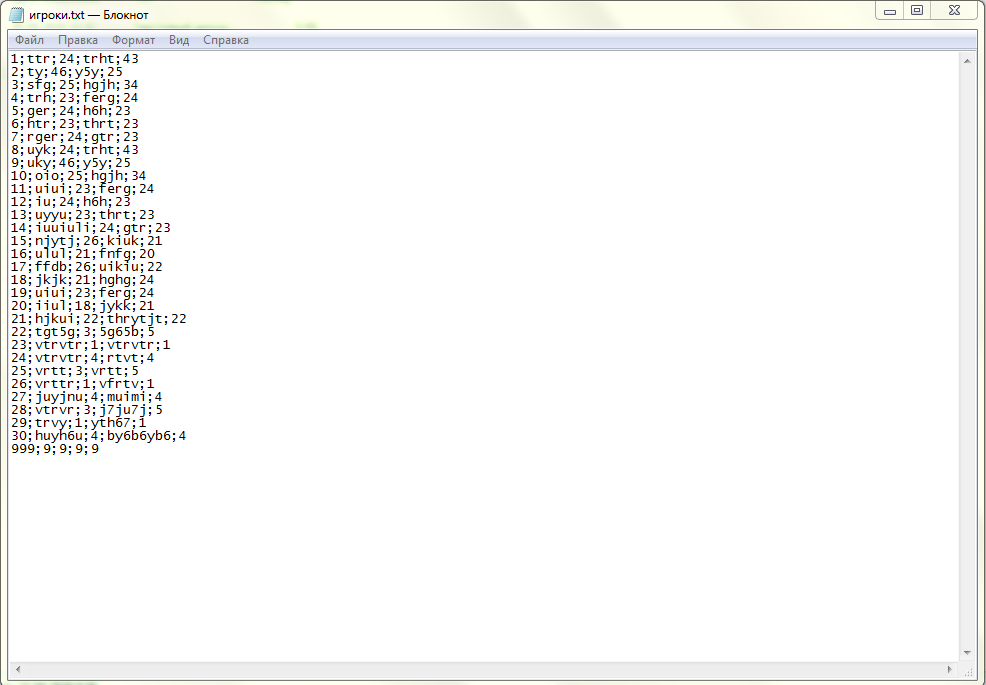


Рисунок 8 – Тест 8 задачи 1

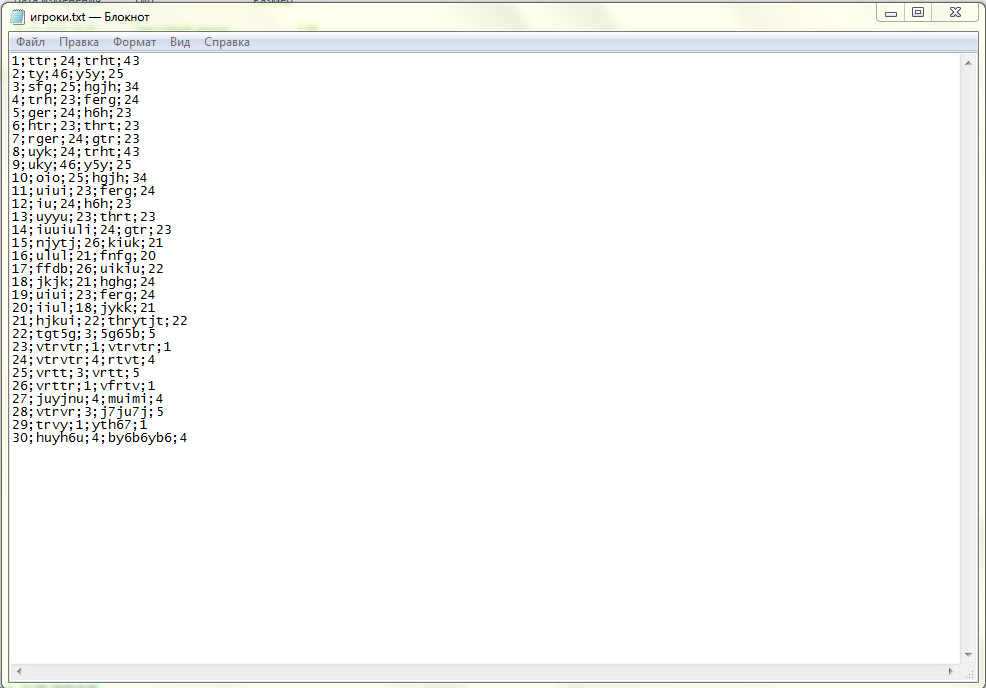


Рисунок 9 – Тест 9 задачи 1

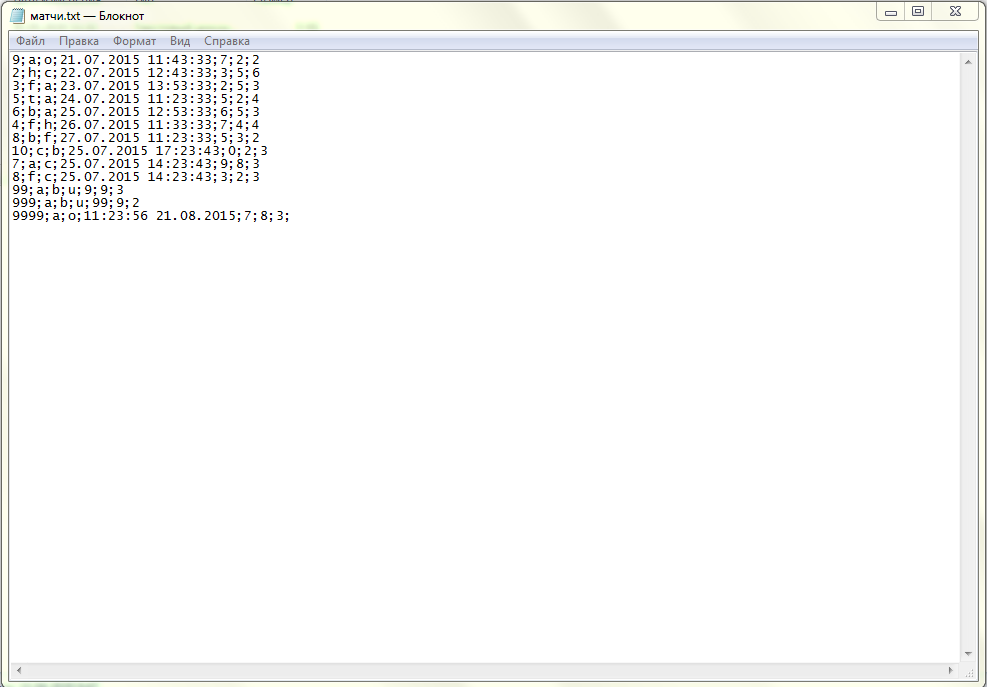


Рисунок 10 – Тест 10 задачи 1

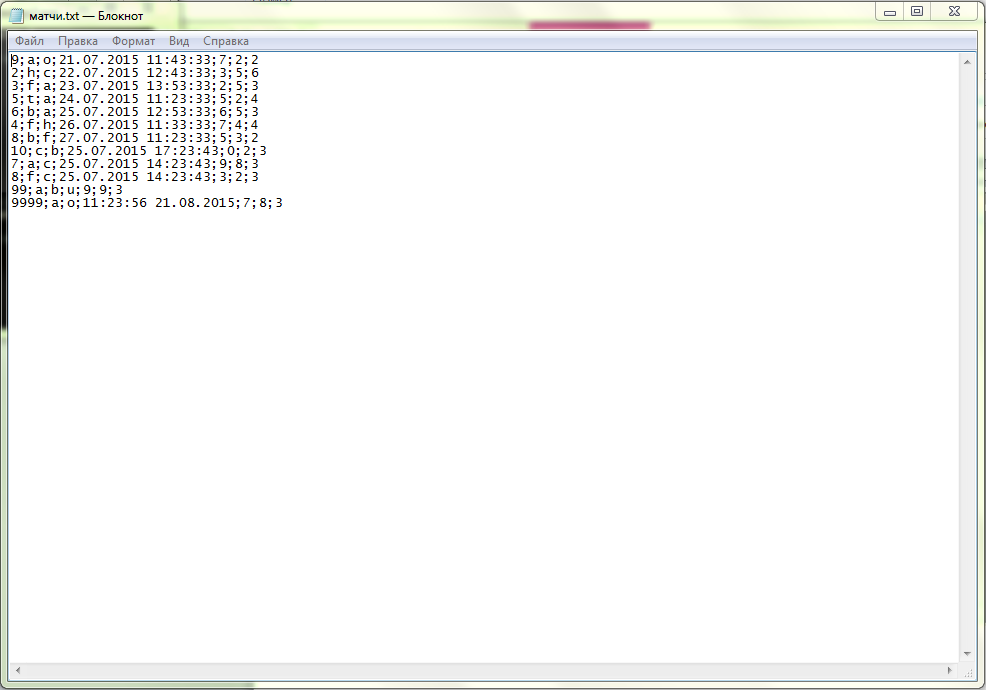


Рисунок 11 – Тест 11 задачи 1

Вывод: в результате выполнения контрольной работы №2 были изучены запросы LINQ к локальным спискам.