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

Основные теоретические положения

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.

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.

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.

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

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).

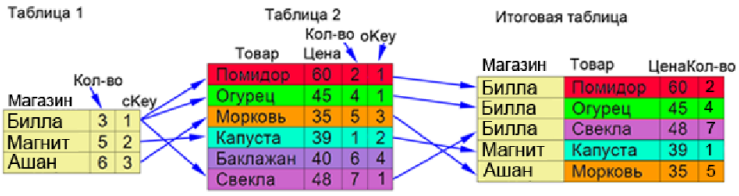

Рисунок 1 – Левое внешнее объединение

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.

Прием в вуз

Требуется разработать программную систему, предназначенную для работников приемной комиссии высшего учебного заведения. Она должна обеспечивать хранение, просмотр и изменение сведений об абитуриентах, а также о расписании экзаменов и консультаций. Результатом работы приемной комиссии должен быть список абитуриентов, зачисленных в институт.

Специальность задается названием кафедры и факультета, на которые поступает абитуриент. При регистрации абитуриенту выдают экзаменационный лист. Абитуриенты на период вступительных экзаменов объединяются в группы. Каждая группа сдает по три экзамена, по которым формируется экзаменационная ведомость. Для каждого потока формируется расписание консультаций и экзаменов по предметам. Медалистам засчитывают все экзамены. Известно количество мест по каждой кафедре. Приемная комиссия по результатам экзаменов должна сформировать списки абитуриентов, зачисленных в институт. Для вычисления проходного балла переписывают абитуриента в список зачисленных из списка результатов экзамена, отсортированного по убыванию оценок, пока не будет достигнуто количество имеющихся мест по выбранной кафедре.

Кафедра	**Специальность**	**Абитуриент**	**Экз. лист**	**Экзамены**
Ключ	Ключ	Ключ	Ключ	Ключ
Название	Название	ФИО	ФИО	Группа
Специальность 1	Проходной балл	Группа	Предмет 1	Экз./Консульт.
Специальность 2	Число мест	Окончил	Оценка 1	Число
		Медаль	Предмет 2	Время
		Специальность	Оценка 2	Аудитория
		Кафедра	Предмет 3	
		Факультет	Оценка 3	

Должны быть созданы обобщенные списки:

– сведения об абитуриентах;

– сведения о расписании;

– сведения об экзаменах;

– массив мест по кафедрам.

Секретарю приемной комиссии могут потребоваться следующие сведения:

– список абитуриентов по группам с учетом кафедры;

– проходной балл по всем специальностям;

– консультация и экзамен по заданному предмету;

– списки зачисленных абитуриентов;

– конкурс на каждую специальность;

– списки абитуриентов, не прошедших по конкурсу;

– кафедры, по которым был недобор абитуриентов.

Необходимо также предусмотреть возможность получения документа, представляющего собой сгруппированные по кафедрам списки абитуриентов, зачисленных в институт, с указанием набранных ими баллов по каждому предмету. Отчет должен содержать проходной балл по каждой кафедре, факультету и по институту в целом, а также количество абитуриентов, поступающих на кафедру и в институт.

Листинг

Program.cs

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

namespace Kontrol._2
{
    class Program
    {
        static List<Enrollee> Enrollees = new List<Enrollee>();
        static List<ExaminationSheet> ExaminationSheets = new List<ExaminationSheet>();
        static List<Exam> Exams = new List<Exam>();
        static List<Specialty> Specialtys = new List<Specialty>();
        static List<Department> Departments = new List<Department>();
```

```
        static string[] PATH = { @"C:\test\Enrollees.txt",
        @"C:\test\ExaminationSheets.txt", @"C:\test\Exams.txt", @"C:\test\Specialtys.txt",
        @"C:\test\Departments.txt" };
         static void Main(string[] args)        {
            foreach (string file in PATH)
                File_Load(file);

            foreach (Enrollee enrollee in Enrollees) //всем медалистам 100 баллов по экзаменам
            {
                if (enrollee.Medal)
                {
                    var index = ExaminationSheets.FindIndex(f => f.Key == enrollee.Key);
                    ExaminationSheets[index].Mark1 = 100;
                    ExaminationSheets[index].Mark2 = 100;
                    ExaminationSheets[index].Mark3 = 100;
                }
            }
            while (true) //для отлова клавиши
            {
                Frame();
                ConsoleKeyInfo key = Console.ReadKey(true);
                Console.Clear();
                switch (key.KeyChar)
                {
                    case '1':
                        var En = Enrollees.Select(e => e); //выборка всех
                        foreach (var en in En)
                            Console.WriteLine(en.ToString());
                        break;
                    case '2':
                        var Ex = Exams.Select(e => e).OrderBy(e => e.Date_Time); //выборка
всех и сортировка по дате
                        foreach (var ex in Ex)
                            Console.WriteLine(ex.ToString());
                        break;
                    case '3':
                        var Ex2 = Exams.Where(e => e.Type == "Экзамен");
                        foreach (var ex in Ex2)
                            Console.WriteLine(ex.ToString());
                        break;
                    case '4':
                        var Dep1 = Departments.Join(Specialtys, d1 => d1.Specialty1, s =>
s.Title, //слияние по 1 спец
                            (d, s) => new { DTitle = d.Title, STitle = s.Title, NOP =
s.NumberOfPlaces });
                        var Dep2 = Departments.Join(Specialtys, d1 => d1.Specialty2, s =>
s.Title, //слияние по 2 спец
                            (d, s) => new { DTitle = d.Title, STitle = s.Title, NOP =
s.NumberOfPlaces });
                        var Dep = Dep1.Concat(Dep2); //конкатенация
                        Dep = Dep.OrderBy(d => d.DTitle); //сортировка
                        foreach (var dep in Dep)
                            Console.WriteLine("{0} - {1}: {2}", dep.DTitle, dep.STitle,
dep.NOP);
                        break;
                    case '5':
                        Console.Clear();
                        bool Query = true;
                        while (Query)
                        {
                            FrameQuery();
                            ConsoleKeyInfo key2 = Console.ReadKey(true);
                            Console.Clear();
                            switch (key2.KeyChar)
                            {
                                case '1':
                                    var EnGroupDep = Enrollees.GroupBy(e => e.Department)
//групировка по кафедре
```

```
                                    .Select(en => new { en.Key, GroupDep = en });
                                foreach (var EGD in EnGroupDep)
                                {
                                    Console.WriteLine("Кафедра: " + EGD.Key); //вывод
кафедры
                                    var EnGroupGr = EGD.GroupDep.GroupBy(e => e.Group)
//группировка по группа
                                        .Select(en => new { en.Key, GroupGr = en });
                                    foreach (var EGG in EnGroupGr)
                                    {
                                        Console.WriteLine("\tГруппа: " + EGG.Key); //вывод
группы
                                        foreach (var egg in EGG.GroupGr)
                                            Console.WriteLine(egg.ToShortString());
//вывод аббитуриентов
                                    }
                                    Console.WriteLine();
                                }
                                break;
                            case '2':
                                var Sp = Specialtys.Select(s => new { s.Title,
s.PassingScore }); //проекция
                                foreach (var s in Sp)
                                    Console.WriteLine("{0} - {1} баллов", s.Title,
s.PassingScore);
                                break;
                            case '3':
                                Console.WriteLine("Введите ключ предмета (1-5):");
                                int Discipline = Convert.ToInt32(Console.ReadLine());
//выбор ключа предмета
                                var Disc = Exams.Where(e => e.Key == Discipline);
//выборка по ключу
                                foreach (var d in Disc)
                                    Console.WriteLine(d.ToString());
                                break;
                            case '4':
                            case '6':
                            case '7':
                                var EnExSheet = Enrollees.Join(ExaminationSheets, e =>
e.Key, es => es.Key, //слияние с абитуриентами
                                    (e, es) => new
                                    {
                                        K = e.Key,
                                        FN = e.FullName,
                                        S = e.Specialty,
                                        Mark = es.Mark1 + es.Mark2 + es.Mark3
                                    });
                                var EnGroup = EnExSheet.GroupBy(e => e.S) //групировка по
специальности
                                    .Select(en => new { en.Key, Group = en });
                                foreach (var ENG in EnGroup)
                                {
                                    Console.WriteLine("Специальность: " + ENG.Key);
                                    var FindS = Specialtys.Find(s => s.Title == ENG.Key);
//поиск специальности
                                    int NOP = FindS.NumberOfPlaces; //количество мест на
специальности
                                    int PS = FindS.PassingScore; //проходной балл
                                    if (key2.KeyChar == '4' || key2.KeyChar == '7')
                                    {
                                        var EnSort = ENG.Group.OrderByDescending(e =>
e.Mark) //сортировка по оценке
                                            .Take(NOP);  //и взятие определённого
количества
                                        if (key2.KeyChar == '4') //список поступивших
                                            foreach (var ES in EnSort)
                                                Console.WriteLine("\t{0} - {1}, Оценка:
{2}", ES.K, ES.FN, ES.Mark);
```

```
                                        if (key2.KeyChar == '7') //сколько недобора
                                            Console.WriteLine(NOP - EnSort.Count());
                                    }
                                    if (key2.KeyChar == '6') //не поступившие по конкурсу
                                    {
                                        var EnSort = ENG.Group.OrderByDescending(e =>
e.Mark) //сортировка по оценке
                                        .Skip(NOP).Where(e => e.Mark >= PS); //пропуск
прошедших и взятие подходящих по баллам
                                        foreach (var ES in EnSort)
                                            Console.WriteLine("\t{0} - {1}, Оценка: {2}",
ES.K, ES.FN, ES.Mark);
                                    }
                                    Console.WriteLine();
                                }
                                break;
                            case '5':
                                var Spec = Specialtys.Select(s => new { s.Title,
s.NumberOfPlaces }); //проекция
                                foreach (var s in Spec)
                                    Console.WriteLine("По специальности {0} конкурс:
{1:F2}", s.Title,
                                    (double)s.NumberOfPlaces / Enrollees.Count(e =>
e.Specialty == s.Title)); //вычисление конкурса
                                break;
                            case '8':
                                Console.Clear();
                                Query = false;
                                break;
                            default:
                                Console.WriteLine("Введите номер существующего запроса!
Нажмите любую клавишу и выберите запрос");
                                Console.ReadKey();
                                Console.Clear();
                                continue;
                        }
                        if (Query) //если не возврат назад
                        {
                            Console.WriteLine("\nЗапрос окончен, нажмите любую клавишу");
                            Console.ReadKey();
                            Console.Clear();
                        }
                        continue;
                    }
                    continue;
                case '6':
                    var EnExSheet2 = Enrollees.Join(ExaminationSheets, e => e.Key, es =>
es.Key, //слияние с абитуриентами
                                    (e, es) => new
                                    {
                                        e.Key,
                                        e.FullName,
                                        e.Specialty,
                                        es.Mark1,
                                        es.Mark2,
                                        es.Mark3,
                                        es.Discipline1,
                                        es.Discipline2,
                                        es.Discipline3
                                    });
                    int GeneralPS = Specialtys.Sum(s => s.PassingScore) /
Specialtys.Count(); //общий проходной балл
                    int GeneralNOP = Specialtys.Sum(s => s.NumberOfPlaces); //общее
количество мест
                    Console.WriteLine("Институт. Общий проходной балл: {0} Общее
количество мест: {1}\n", GeneralPS, GeneralNOP);
                    var EnGroup2 = EnExSheet2.GroupBy(e => e.Specialty) //групировка по
специальности
```

```
                            .Select(en => new { en.Key, Group = en });
                        foreach (var ENG in EnGroup2)
                        {
                            var FindS = Specialtys.Find(s => s.Title == ENG.Key); //поиск специальности
                            int NOP = FindS.NumberOfPlaces; //количество мест на специальности
                            int PS = FindS.PassingScore; //проходной балл
                            Console.WriteLine("{0} - проходной балл: {1}, количество мест: {2}", ENG.Key, PS, NOP);
                            var EnSort = ENG.Group.OrderByDescending(e => e.Mark1 + e.Mark2 + e.Mark3) //сортировка по оценке
                                .Take(NOP);  //и взятие определённого количества
                            foreach (var ES in EnSort)
                                Console.WriteLine("\t{0} - {1} Предмет: {2} ({3}), Предмет: {4} ({5}), Предмет: {6} ({7}), Общий балл: {8}"
                                    , ES.Key, ES.FullName, ES.Discipline1, ES.Mark1, ES.Discipline2, ES.Mark2, ES.Discipline3, ES.Mark3,
                                    ES.Mark1 + ES.Mark2 + ES.Mark3);
                            Console.WriteLine();
                        }
                        break;
                    case '7':
                        Environment.Exit(0);
                        break;
                    default:
                        Console.WriteLine("Введите номер существующей задачи! Нажмите любую клавишу и выберите задачу");
                        Console.ReadKey();
                        Console.Clear();
                        continue;
                }
                Console.WriteLine("\nЗапрос окончен, нажмите любую клавишу");
                Console.ReadKey();
                Console.Clear();
                continue;
            }
        }
        static void File_Load(string path)        //загрузка данных
        {
            StreamReader FileIn = new StreamReader(path, Encoding.UTF8);
            string str;
            while ((str = FileIn.ReadLine()) != null)  //пока не конец файла
            {
                string[] ms = str.Split(';');
                if (path == "C:\test\Enrollees.txt")
                    Enrollees.Add(new Enrollee(Convert.ToInt32(ms[0]), ms[1], Convert.ToInt32(ms[2]), ms[3],
                        Convert.ToBoolean(Convert.ToInt32(ms[4])), ms[5], ms[6], ms[7]));
                else if (path == "C:\test\ExaminationSheets.txt")
                    ExaminationSheets.Add(new ExaminationSheet(Convert.ToInt32(ms[0]), ms[1], Convert.ToInt32(ms[2]), Convert.ToInt32(ms[3]),
                        Convert.ToInt32(ms[4]), Convert.ToInt32(ms[5]), Convert.ToInt32(ms[6]), Convert.ToInt32(ms[7])));
                else if (path == "C:\test\Exams.txt")
                    Exams.Add(new Exam(Convert.ToInt32(ms[0]), Convert.ToInt32(ms[1]), ms[2], DateTime.Parse(ms[3]), Convert.ToInt32(ms[4])));
                else if (path == "C:\test\Specialtys.txt")
                    Specialtys.Add(new Specialty(Convert.ToInt32(ms[0]), ms[1], Convert.ToInt32(ms[2]), Convert.ToInt32(ms[3])));
                else if (path == "C:\test\Departments.txt")
                    Departments.Add(new Department(Convert.ToInt32(ms[0]), ms[1], ms[2], ms[3]));
            }
        }
        static void Frame() //метод для выбора списка задач
        {
            Console.WriteLine("\tВыберете номер задачи\n");
            Console.WriteLine("1 - Получить список аббитуриентов");
```

```
            Console.WriteLine("2 - Получить расписание экзаменов");
            Console.WriteLine("3 - Получить сведения об экзаменах");
            Console.WriteLine("4 - Получить информацию по местам на кафедрах");
            Console.WriteLine("5 - Запросы");
            Console.WriteLine("6 - Отчёт");
            Console.WriteLine("\n7 - Выход из программы");
        }
        static void FrameQuery() //метод для выбора списка запросов
        {
            Console.WriteLine("\tВыберете номер запроса\n");
            Console.WriteLine("1 - Получить список аббитуриентов по группам с учётом
кафедры");
            Console.WriteLine("2 - Получить проходные баллы по специальностям");
            Console.WriteLine("3 - Получить консультацию и экзамен по заданному предмету");
            Console.WriteLine("4 - Получить список зачисленных аббитуриентов");
            Console.WriteLine("5 - Получить конкурс по специальностям");
            Console.WriteLine("6 - Получить список аббитуриентов не прошедших по конкурсу");
            Console.WriteLine("7 - Получить список кафедр по которым был недобор студентов");
            Console.WriteLine("\n8 - Назад");
        }

    }
}
```

Department.cs

```
namespace Kontrol._2
{
    class Department
    {
        public int Key { get; set; }                //Ключ
        public string Title { get; set; }           //Название
        public string Specialty1 { get; set; }      //Специальность 1
        public string Specialty2 { get; set; }      //Специальность 2
        public Department(int K, string T, string S1, string S2)
        {
            Key = K; Title = T; Specialty1 = S1; Specialty2 = S2;
        }
    }
}
```

Specialty.cs

```
namespace Kontrol._2
{
    class Specialty
    {
        public int Key { get; set; }                //Ключ
        public string Title { get; set; }           //Название
        public int PassingScore { get; set; }       //Проходной балл
        public int NumberOfPlaces { get; set; }     //Число мест
        public Specialty(int K, string T, int PS, int NOP)
        {
            Key = K; Title = T; PassingScore = PS; NumberOfPlaces = NOP;
        }
    }
}
```

Enrollee.cs

```
namespace Kontrol._2
{
    class Enrollee
    {
        public int Key { get; set; }            //Ключ
        public string FullName { get; set; }    //ФИО
        public int Group { get; set; }          //Группа
        public string Graduated { get; set; }   //Что закончил
```

```
        public bool Medal { get; set; }         //Наличие медали
        public string Specialty { get; set; }   //Специальность
        public string Department { get; set; }  //Кафедра
        public string Faculty { get; set; }     //Факультет
        public Enrollee(int K, string FN, int Gro, string Gra, bool M, string S, string D,
string F)
        {
            Key = K; FullName = FN; Group = Gro; Graduated = Gra; Medal = M; Specialty = S;
Department = D; Faculty = F;
        }
        public override string ToString()
        {
            return string.Format("{0} - {1}\n\tГруппа: {2}, Окончил: {3}, Медаль: {4}, " +
                "Специальность: {5}, Кафедра: {6}, Факультет: {7}", Key, FullName, Group,
Graduated,
                (Medal) ? "есть" : "нет", Specialty, Department, Faculty);
        }
        public string ToShortString()
        {
            return string.Format("{0} - {1}\n\t Окончил: {2}, Медаль: {3}, Специальность:
{4}",
                Key, FullName, Graduated, (Medal) ? "есть" : "нет", Specialty);
        }
    }
}
```

ExaminationSheet.cs

```
namespace Kontrol._2
{
    class ExaminationSheet
    {
        public int Key { get; set; }               //Ключ
        public string FullName { get; set; }       //ФИО
        public int Discipline1 { get; set; }       //Предмет 1 ключ
        public int Mark1 { get; set; }        //Оценка по предмету 1
        public int Discipline2 { get; set; }       //Предмет 2 ключ
        public int Mark2 { get; set; }        //Оценка по предмету 2
        public int Discipline3 { get; set; }       //Предмет 3 ключ
        public int Mark3 { get; set; }        //Оценка по предмету 3
        public ExaminationSheet(int K, string FN, int D1, int M1, int D2, int M2, int D3, int
M3)
        {
            Key = K; FullName = FN; Discipline1 = D1; Mark1 = M1; Discipline2 = D2; Mark2 =
M2; Discipline3 = D3; Mark3 = M3;
        }
    }
}
```

Exam.cs

```
using System;

namespace Kontrol._2
{
    class Exam
    {
        public int Key { get; set; }            //Ключ
        public int Group { get; set; }          //Группа
        public string Type { get; set; }        //Экзамен или Консультация
        public DateTime Date_Time { get; set; } //Дата и Время
        public int LectureRoom { get; set; }    //Номер аудитории
        public Exam(int K, int G, string T, DateTime DT, int LR)
        {
            Key = K; Group = G; Type = T; Date_Time = DT; LectureRoom = LR;
        }
        public override string ToString()
        {
```

```
            return string.Format("{0} - {1}, Группа: {2}, Дата: {3}, Время: {4}, " +
                "Аудитория: {5}", Key, Type, Group, Date_Time.ToString("dd.MM.yyyy"),
                Date_Time.ToString("hh:mm"), LectureRoom);
        }
    }
}
```

Тесты

Результаты тестирования представлены на рисунках 2 – 5.

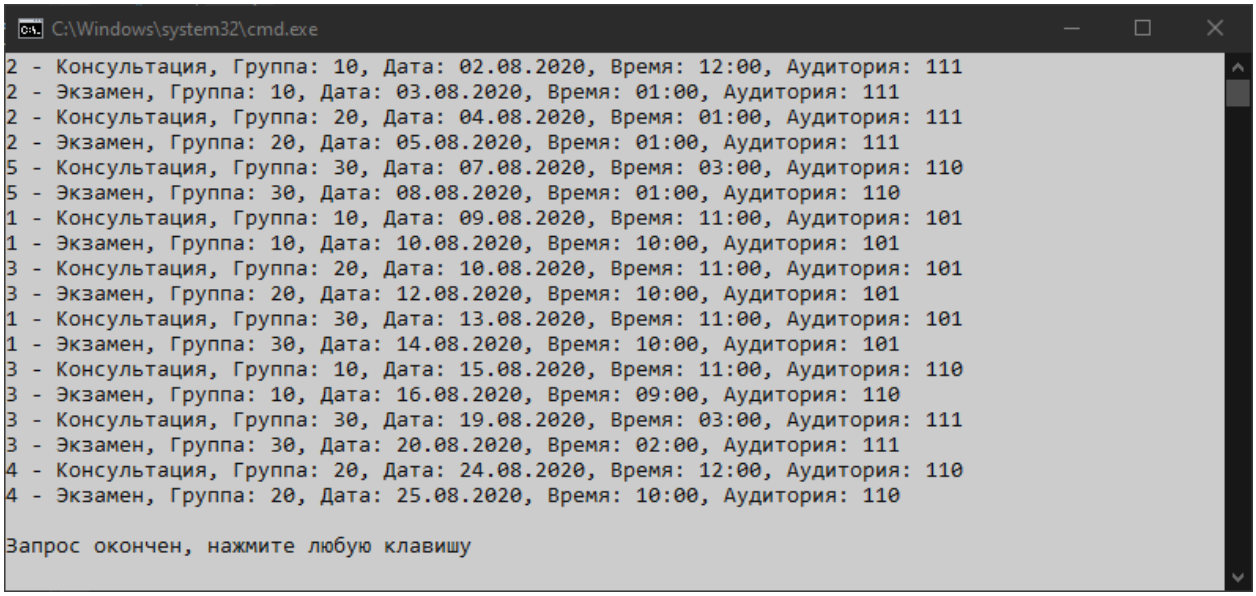

Рисунок 2 – Тест одной из задачи

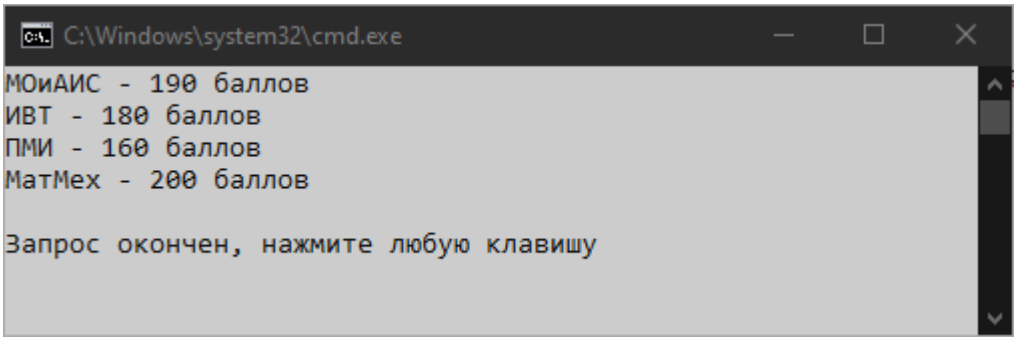

Рисунок 3 – Тест 1 запроса

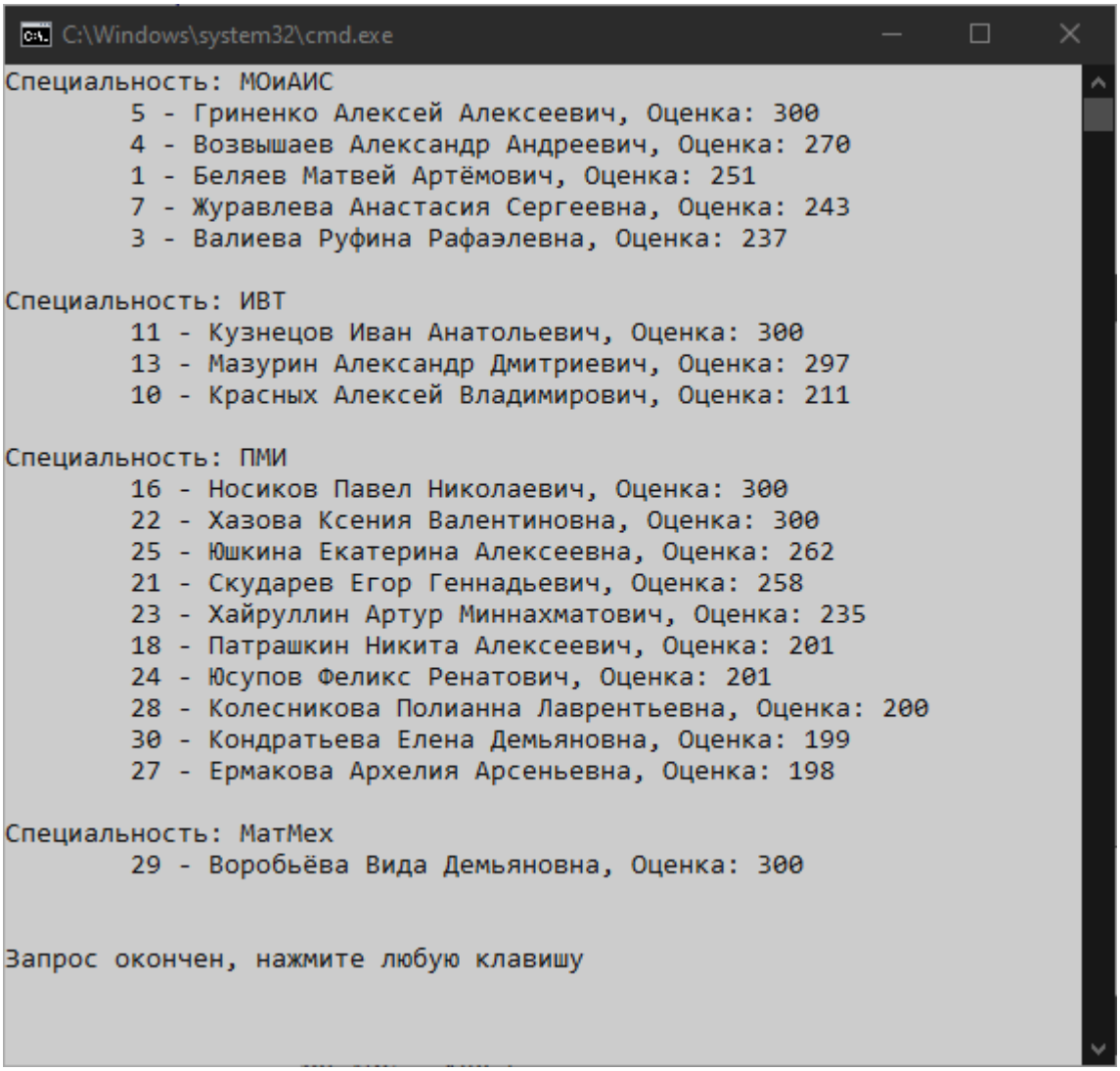

Рисунок 4 – Тест 2 запроса

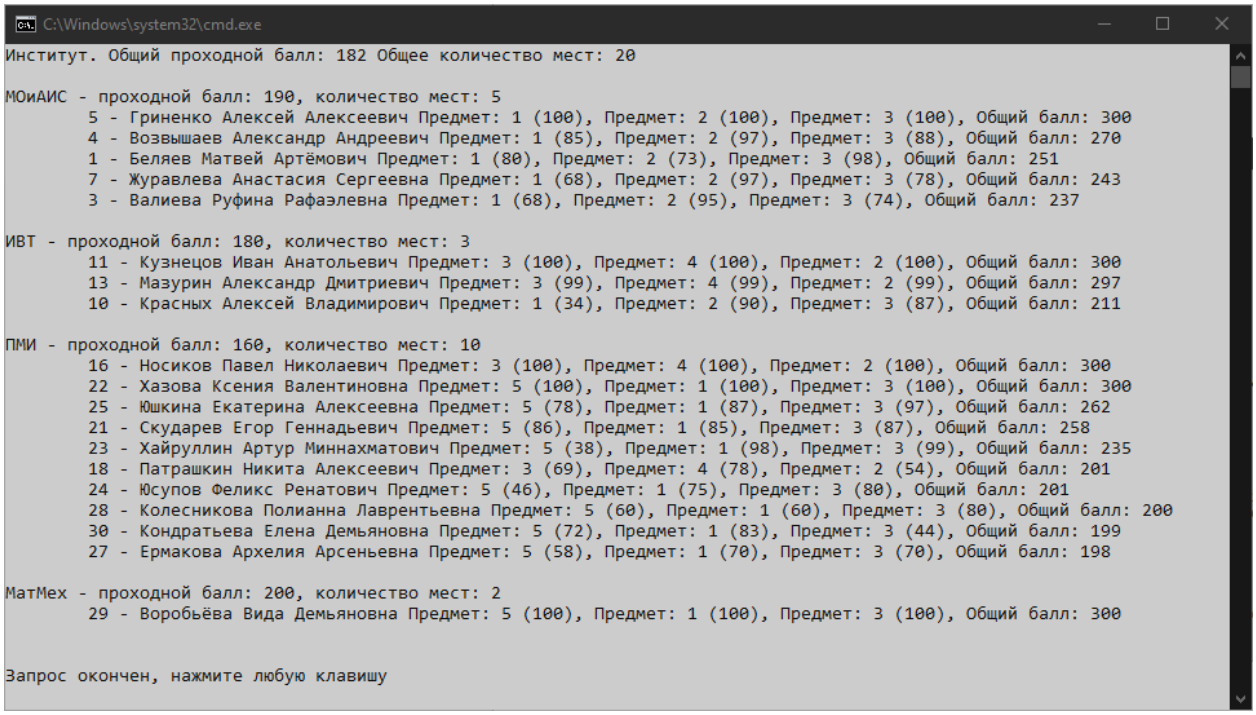

Рисунок 5 – Тест отчёта

Выводы: разработал в среде Microsoft Visual Studio консольное приложение для работы с локальными обобщёнными списками.