Министерство образования и науки Российской Федерации Федеральное государственное автономное образовательное учреждение высшего профессионального образования

**«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»**

Институт радиоэлектроники и информационных технологий – РтФ

Кафедра «Информационных технологий»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2**

Студенка группы:

РИ-240004

Патракова Е.C.

Преподаватель:

Быков Е.А.

Екатеринбург

2016

Содержание:

1. Введение3
2. Описание задания3
3. Описание библиотек, классов, основных методов5
   1. Использованные готовые классы5
   2. Созданные классы6
4. Результаты работы9
5. Анализ полученных данных12
6. Код программы13
7. Выводы23
8. **Введение:**

Целью данной лабораторной работы является написание простого консольного приложения для изучения методов работы со строками, списками и словарем; ознакомления с LINQ запросами и лямбда-выражениями.

В ходе выполнения работы подлежит решению следующая задача:

Написание функционального консольного приложения для решения нескольких задач из различных предметных областей и с привлечением различных средств и особенностей языка программирования.

1. **Описание задания:**
   1. Выделение части строки по двум разделителям используя класс String. Пользователь вводит некоторое сообщение. Далее пользователь вводит часть, с которой он хочет начинать вырезку предложения и часть, определяющую конец предложения.
   2. Нахождение суммы и среднего значения, мин. и макс. Целочисленного списка. Данное задание должно быть описано в отдельный класс с функциями нахождения суммы, среднего значения, минимума и максимума. Должен быть реализован один конструктор с входным параметром строки. Пример строки: «11,54,34,12,43,543,6,16». Попадая в конструктор необходимо разбить данную строку по запятой, в результате получится список (массив), его необходимо преобразовать в целочисленное множество и поместить в глобальную переменную (поле).
   3. Необходимо придумать любую таблицу с инициализированными данными. Данный блок заданий нужно выполнять в трех разных вариантах (одновременно):

* итеративных методов, то есть методом перебора проверяем условия отбора;
* • лямбда выражений;
* • LINQ-запросов.

Необходимо создать класс (сущности), который будет характеризовать выбранную тему. То есть создать класс, в котором будет описано не менее 3 полей.

* + 1. **Фильтрация по строке**

Реализовать 2 метода:

* формировать список элементов, фильтруя по некоторому string полю по следующему правилу: пользователь прописывает произвольный символ, фильтр должен добавить в список только те элементы, у которых содержимое поля будет начинаться с символа, меньше заданного пользователем (по алфавиту);
* искать элемент, строковое поле которого содержит часть слова, написанного пользователем. Пример: пользователь написал часть слова «ал» метод должен найти все элементы в определенном поле которого присутствует часть слова «ал» (В качестве лямбда выражения за основу взять string.Compare).
  + 1. **Нахождение элементов по заданному промежутку**

Нахождение элементов числовое поле которых соответствует введенным данным пользователя. Например, пользователь задает начальное значение, например 18, и конечное значения поиска, например 19. Результат будет список элементов данные поля которых соответствует заданным.

* + 1. **Сортировка**

Создать двойную сортировку. Например, изначально делать сортировка по строковому полю, далее сортировку по числовому полю.

* + 1. **Сложная фильтрация**

Реализовать фильтрацию элементов по следующему критерию:

* строковое поле должно содержать часть слова введенных пользователем;
* числовое поле должно без остатка делиться на 7 и 3•

1. **Описание библиотек, классов и основных методов:**

#### 3.1 Использованные готовые классы

* Класс **Console** пространства имен System

Предоставляет стандартные потоки для консольных приложений: входной, выходной и поток сообщений об ошибках.

Метод WriteLine() для вывода текста на консоль.

Метод ReadLine() для считывания текста с консоли

Метод ReadKey() для посимвольного считывания.

Свойство KeyChar возвращает или задает символ, соответствующий нажатой клавише.

* Структура **Int** пространства имен System

Представляет целое число со знаком.

Метод Parse(String) для преобразования считанной с консоли входной строки к формату целого числа.

* Структура **Double** пространства имен System

Представляет число двойной точности с плавающей точкой.

Метод Parse(String) для преобразования считанной с консоли входной строки к формату числа с плавающей точкой.

ToString(double) для преобразования числа двойной точности с плавающей точкой к формату строки;

* Структура **DateTime** пространства имен System:

Представляет текущее время, обычно выраженное как дата и время суток.

Метод Parse(String) для преобразования считанной с консоли входной строки к формату даты.

Метод IsLeapYear(Int), позволяющий определить, является ли заданный год високосным.

Свойство Now, возвращающее текущие дату и время.

* Структура **TimeSpan** пространства имен System

Представляет интервал времени.

В программе создается экземпляр структуры для хранения интервала между двумя датами, выраженного в днях (свойство Days);

* Класс **String** пространства имен System

Представляет текст как последовательность знаков Юникода.

Метод Split(Char[], StringSplitOptions)для разбивания строки, включающей разделитель (пробел), на подстроки без включения в массив подстрок самих разделяющих символов.

Метод Replace(String, String) для замены одной подстроки в строке другой.

Метод Substring(int startIndex, int Lenght) выделяет подстроку из строки начиная с указанного индекса некоторой длины.

Метод CompareTo(String) сравнивает данный экземпляр с заданным объектом String и показывает, расположен ли данный экземпляр перед, после или на той же позиции в порядке сортировки, что и заданная строка.

Метод ToLower() переводит строку в нижний регистр. Возвращает копию строки.

Метод Contains(string) возвращает значение, указывающее, содержит ли указанная строка значение подстроки переданной в качестве параметра.

Свойство Length возвращает число знаков в текущем объекте String.

* Класс List <T> пространства имен [System.Collections.Generic](https://msdn.microsoft.com/ru-ru/library/system.collections.generic(v=vs.110).aspx)

Представляет строго типизированный список объектов, доступных по индексу. Поддерживает методы для поиска по списку, выполнения сортировки и других операций со списками.

Метод Add(T) добавляет объект в конец очереди.

Метод IndexOf(string ) возвращает индекс первого вхождения в строку аргумента.

Метод Sum() для получения суммы всех элементов листа.

Метод Average() для получения среднего арифметического элементов листа.

Метод Min() для получения минимального элемента листа.

Метод Max() для получения максимального значения листа.

Свойство Count получает число элементов, содержащихся в листе.

#### 3.2. Созданные классы

1. Класс Program пространства имен LR2

Включает в себя cтатический метод Main(), являющийся точкой входа в программу и обеспечивающий диалог программы и пользователя.

Метод EXIT() обеспечивающий диалог о выходе или продолжении работы. Прототип: *public bool* EXIT();

Метод List() осуществляющий корректное считывание строки целых чисел. Прототип: static void List();

1. Класс Text пространства имен LR2

Включает в себя закрытые поля для хранения исходной строки, начала вырезки и конца вырезки, конструктор по умолчанию и методы CutLine(), обеспечивающий корректный ввод строки, начала и конца вырезки, и метод Cut() вырезающий строк, выводящий её на экран.

Прототип метода CutLine(): *public void* CutLine();

Прототип метода Cut(): *public void* Cut();

1. Класс IntList пространства имен LR2

Представляет собой контейнер для целых чисел со встроенным анализатором. Включает в себя закрытое поле типа List<int>. Класс так же содержит конструктор, принимающий строку, и методы Menu() для выбора действия.

Метод Sum(), считающий сумму чисел в списке.

Прототип private void Sum();

Метод Average(), считающий среднее арифметическое элементов списка.

Прототип private void Average();

Метод Minimum(), который находит минимальное число с списке. Прототип private void Minimum();

Метод Maximum(), который находит максимальное число с списке. Прототип private void Maximun();

1. Класс ProgramException пространства имен LR2

Наследник SystemException.

Включает в себя конструктор, принимающий строку и выводящий сообщение на консоль.

1. Класс Film пространства имен LR2

Класс описывает фильм. Включает в себя закрытые поля типа string для названия и жанра, поле типа DateTime для даты премьеры, целое поле для рейтинга. Два конструктора один для инициализации всех полей, другой для инициализации объектом того же типа Film.

Класс содержит шесть открытых методов меню.

Метод getList() возвращает лист фильмов. Прототип: public static List<Film> getList().

Метод FiltreByName() вызывающий методы для фильтрации названий фильмов по букве. Прототип: public void FiltreByName().

Метод FiltreByPart() вызывающий методы для фильтрации фильмов по части названия. Прототип: public void FiltreByPart().

Метод FindElements () вызывающий методы для отбора фильмов по промежутку лет. Прототип: public void FiltreElements().

Метод Sort() вызывающий методы для сортировки фильмов по двум полям по возрастанию по названию и рейтингу. Прототип: public void Sort().

Метод DoubleSort() вызывающий методы для двойной сортировки фильмов по части названия и рейтингу. Прототип: public void DoubleSort ().

Класс содержит 16 скрытых методов:

Методы LINQFiltreByPart(), LambdaFiltreByPart(), IterFiltereByPart() выполняют одно и тоже, а именно фильтрацию фильмов по части названия. Выводятся только те фильмы, названия которых содержат часть.

Методы LINQFiltreByName(),LambdaFiltreByName(),IterFiltreByName(), выводят фильмы, первые буквы названий которых раньше по алфавиту, чем введенная для фильтрации.

Методы LINQFiltreByDate(), LambdaFiltreByDate(), IterFiltreByDate() методы отбирающие фильмы по промежутку лет.

Методы LINQSort(), LambdaSort(), IterSort() выполняют сортировку по названию по алфавиту, а если названия одинаковы, то по рейтингу( высокий выводится раньше).

Методы LINQDoubleFiltre(), LambdaDoubleFiltre (), IterDoubleFiltre () двойная фильтрация по части, содержащейся в названии и рейтингу, который нацело делится и на 3, и на 7.

Метод PrintFiltredList() выводит на консоль отфильтрованный список по ключу.

Прототипы:

private void LINQFiltreByPart(string part);

private void LambdaFiltreByPart(string part);

private void IterFiltreByPart(string part);

private void LINQFiltreByName(char key);

private void LambdaFiltreByName(char key);

private void IterFiltreByName(char key);

private void LINQFiltreByDate(int startYear, int endYear);

private void LambdaFiltreByDate(int startYear, int endYear);

private void IterFiltreByDate(int startYear, int endYear);

private void LINQSort();

private void LambdaSort();

private void IterSort();

private void LINQDoubleFiltre(string part);

private void LambdaDoubleFiltre(string part);

private void IterDoubleFiltre(string part);

private void PrintFiltredList(ref List<Film> filteredList, string key)

1. **Результаты работы программ:**
   1. Меню:

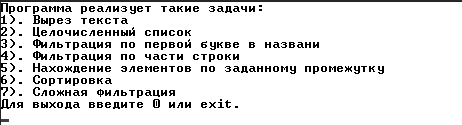


Рис. 1. Меню приложения

* 1. Вырезка строки:

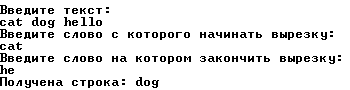


Рис. 2. Нормальный ход программы

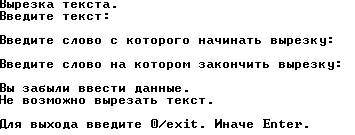


Рис. 3. Обработка пустой строки \ строки с пробелами

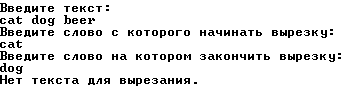


Рис. 4. Ошибка, если два слова стоят рядом.

* 1. Целые числа:

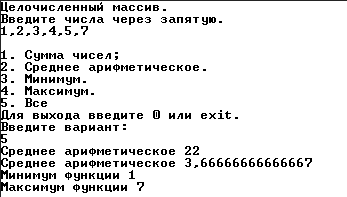
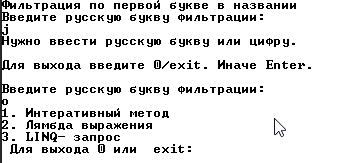


Рис. 5. Нормальный ход программы

* 1. Фильтрации по первой букве в названии



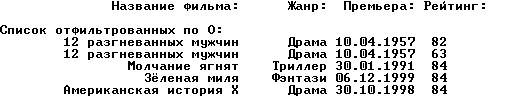


Рис. 6. Нормальный ход программы

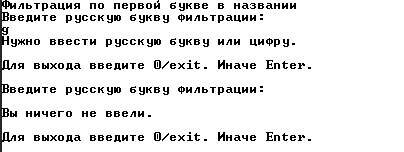
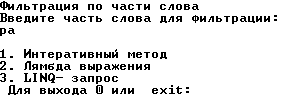


Рис. 7. Ошибка. Неправильный ввод

* 1. Фильтрация по части строки



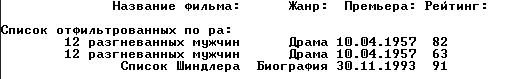


Рис. 8. Нормальный ход программы

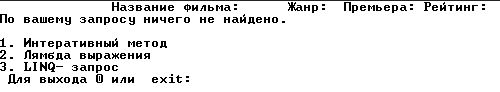


Рис. 9.Ошибка. Ничего не найдено

* 1. Нахождение элементов по заданному промежутку

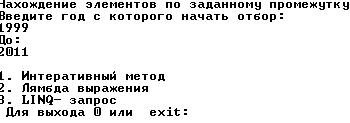


Рис. 10. Меню

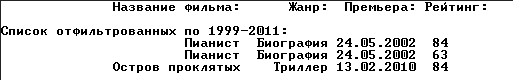


Рис. 11. Фильтрация по промежутку лет

* 1. Сортировка:

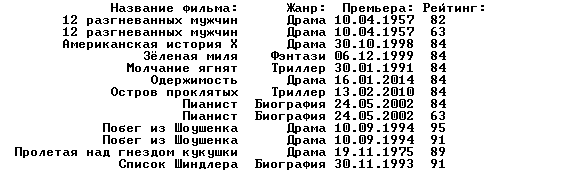
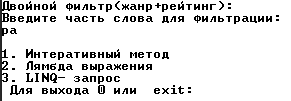


Рис. 12. Сортировка по алфавиту.

* 1. Двойная фильтрация:

1. 

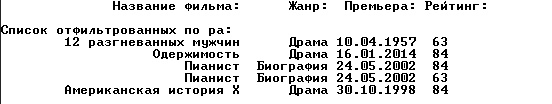


Рис. 13. Двойная сортировка.

1. **Анализ полученных данных:**

В результате выполнения лабораторной работы было разработано многофункциональное консольное приложение, реализующее решение математических и логических задач, предлагаемых пользователю на выбор. Для осуществления диалога программы и пользователя в приложении реализовано главное меню, позволяющее выбрать одну из решаемых им подзадач для выполнения.

1. **Код программы:**
2. Функция Main():

using System;

namespace LR2

{

class Program

{

static void Main(string[] args)

{

string task;

while (true)

{

Console.Clear();

Console.WriteLine(

"Программа реализует такие задачи:\n1). Вырез текста\n2). Целочисленный список"+

"\n3). Фильтрация по первой букве в названи\n4). Фильтрация по части строки\n"+

"5). Нахождение элементов по заданному промежутку\n6). Сортировка\n7). Сложная фильтрация\nДля выхода введите 0 или exit.");

task = Console.ReadLine();

switch (task)

{

case "0":

case "exit":

return;

case "1":

Console.WriteLine("Вырезка текста.");

while (true)

{

Text text = new Text();

text.CutLine();

if (EXIT())

break;

}

break;

case "2":

Console.WriteLine("Целочисленный массив.\nВведите числа через запятую.");

while (true)

{

List();

if (EXIT())

break;

}

break;

case "3":

Console.WriteLine("Фильтрация по первой букве в названии");

while (true)

{

Film film=new Film();

film.FiltreByName();

if (EXIT())

break;

}

break;

case "4":

Console.WriteLine("Фильтрация по части слова");

while (true)

{

Film film = new Film();

film.FiltreByPart();

if (EXIT())

break;

}

break;

case "5":

Console.WriteLine("Нахождение элементов по заданному промежутку");

while (true)

{

Film film = new Film();

film.FindElements();

if (EXIT())

break;

}

break;

case "6":

Console.WriteLine("Сортировка");

while (true)

{

Film film = new Film();

film.Sort();

if (EXIT())

break;

}

break;

case "7":

Console.WriteLine("Двойной фильтр(жанр+рейтинг):");

while (true)

{

Film film = new Film();

film.DoubleFiltre();

if (EXIT())

break;

}

break;

}

}

}

static bool EXIT()

{

string exit;

Console.WriteLine("\nДля выхода введите 0/exit. Иначе Enter.");

exit = Console.ReadLine();

if (exit == "0" || exit == "exit")

return true;

return false;

}

static void List()

{

try

{

string line = Console.ReadLine().Replace('.', ',').Trim(' ');

if (line.Length == 0)

{

throw new ProgramException("\nВы ничего не ввели.");

}

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

{

if ((line[i] < '0' || line[i] > '9') && line[i] != ',')

{

throw new ProgramException("\nОшибка ввода");

}

}

IntList intList = new IntList(line);

intList.Menu();

}

catch (Exception e)

{ }

}

}

}

1. Класс Text

public class Text

{

private string text;

private string start;

private string stop;

public void CutLine()

{

try

{

Console.WriteLine("Введите текст:");

text = Console.ReadLine().ToLower();

Console.WriteLine("Введите слово с которого начинать вырезку: ");

start = Console.ReadLine().ToLower();

Console.WriteLine("Введите слово на котором закончить вырезку: ");

stop = Console.ReadLine().ToLower();

if (text.Length == 0 || start.Length == 0 || stop.Length == 0)

{

Console.WriteLine("Вы забыли ввести данные.");

}

if (Equals(text, start) || Equals(text, stop) || Equals(start, stop))

{

throw new ProgramException("Не возможно вырезать текст.");

}

}

catch (ArgumentNullException e)

{

Console.WriteLine("Вы забыли ввести данные.");

}

catch

{

}

Cut();

}

private void Cut()

{

try

{

int begin = text.IndexOf(start);

int end = text.IndexOf(stop);

int startIndex = begin + start.Length;

if (text[start.Length] == ' ')

startIndex++;

if (startIndex == end)

{

throw new ProgramException("Нет текста для вырезания.");

}

int endIndex = end - start.Length - 1;

var result = text.Substring(startIndex, endIndex);

Console.WriteLine("Получена строка: {0}", result);

}

catch (ArgumentOutOfRangeException e)

{

Console.WriteLine("Не удалось вырезать текст, потому что " + e.Message);

}

catch

{

}

}

}

1. Класс IntList

public class IntList

{

List<int> list;

public IntList(string line)

{

try

{

string[] numbers = line.Split(new char[] {' ', ',', ' '}, StringSplitOptions.RemoveEmptyEntries);

list = new List<int>(numbers.Length);

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

list.Add(int.Parse(numbers[i]));

}

catch (Exception e)

{

Console.WriteLine("Ошибка во время выполнения.");

}

}

public void Menu()

{

while (true)

{

Console.WriteLine("\n1. Сумма чисел; \n2. Среднее арифметическое.\n3. Минимум. \n4. Максимум." +

"\nДля выхода введите 0 или exit.\nВведите вариант: ");

string task = Console.ReadLine();

switch (task)

{

case "0":

case "exit":

return;

case "1":

this.Sum();

break;

case "2":

this.Average();

break;

case "3":

this.Minimum();

break;

case "4":

this.Maximum();

break;

}

}

}

private void Sum()

{

Console.WriteLine("Среднее арифметическое {0}", list.Sum());

}

private void Average()

{

Console.WriteLine("Среднее арифметическое {0}", list.Average());

}

private void Minimum()

{

Console.WriteLine("Минимум функции {0}", list.Min());

}

private void Maximum()

{

Console.WriteLine("Максимум функции {0}", list.Max());

}

}

1. Класс Film

using System;

using System.Collections.Generic;

using System.Linq;

class Film

{

private string title { get; set; }

private string genre { get; set; }

private DateTime date { get; set; }

private int raiting { get; set; }

public Film(string t = "Побег из Шоушенка", string g = "Драма", string d = "10 9 1994", int r = 91)

{

genre = g;

title = t;

date = DateTime.Parse(d);

raiting = r;

}

public Film(Film some)

{

genre = some.genre;

title = some.title;

date = some.date;

raiting = some.raiting;

}

public void FiltreByName()

{

try

{

Console.WriteLine("Введите русскую букву фильтрации:");

char key = Char.ToUpper(Console.ReadKey().KeyChar);

if (key=='\r')

{

throw new ProgramException("\nВы ничего не ввели.");

}

if ((key<'А' || key>'я') &&( key<'0' || key>'9'))

{

throw new ProgramException("\nНужно ввести русскую букву или цифру.");

}

while (true)

{

Console.WriteLine(

"\n1. Интеративный метод\n2. Лямбда выражения\n3. LINQ- запрос\n Для выхода 0 или exit:");

switch (Console.ReadLine())

{

case "0":

case "exit":

return;

case "1":

IterFiltreByName(key);

break;

case "2":

LambdaFiltreByName(key);

break;

case "3":

LINQFiltreByName(key);

break;

}

}

}

catch{}

}

public void FiltreByPart()

{

try

{

Console.WriteLine("Введите часть слова для фильтрации:");

string part = Console.ReadLine().ToLower();

if (part == "" || part == " ")

{

throw new ProgramException("\nВы ничего не ввели.");

}

while (true)

{

Console.WriteLine(

"\n1. Интеративный метод\n2. Лямбда выражения\n3. LINQ- запрос\n Для выхода 0 или exit:");

switch (Console.ReadLine())

{

case "0":

case "exit":

return;

case "1":

IterFiltreByPart(part);

break;

case "2":

LambdaFiltreByPart(part);

break;

case "3":

LINQFiltreByPart(part);

break;

}

}

}

catch{}

}

public void FindElements()

{

Console.WriteLine("Введите год с которого начать отбор:");

int startYear = Int32.Parse(Console.ReadLine());

Console.WriteLine("До:");

int endYear = Int32.Parse(Console.ReadLine());

if (1940 > startYear && startYear > 2016 || endYear < 1940 && endYear > 2016 || startYear == null ||

endYear == null)

{

Console.WriteLine("Не верно задан промежуток");

return;

}

if (startYear > endYear)

{

int temp = startYear;

startYear = endYear;

endYear = temp;

}

while (true)

{

Console.WriteLine("\n1. Интеративный метод\n2. Лямбда выражения\n3. LINQ- запрос\n Для выхода 0 или exit:");

switch (Console.ReadLine())

{

case "0":

case "exit":

return;

case "1":

IterFiltreByPart(startYear, endYear);

break;

case "2":

LambdaFiltreByPart(startYear, endYear);

break;

case "3":

LINQFiltreByPart(startYear, endYear);

break;

}

}

}

public void Sort()

{

Console.Clear();

while (true)

{

Console.WriteLine("\n1. Интеративный метод\n2. Лямбда выражения\n3. LINQ- запрос\n Для выхода 0 или exit:");

switch (Console.ReadLine())

{

case "0":

case "exit":

return;

case "1":

IterSort();

break;

case "2":

LambdaSort();

break;

case "3":

LINQSort();

break;

}

}

}

public void DoubleFiltre()

{

try

{

Console.WriteLine("Введите часть слова для фильтрации:");

string part = Console.ReadLine().ToLower();

if (part == "" || part==" ")

{

throw new ProgramException("\nВы ничего не ввели.");

}

while (true)

{

Console.WriteLine(

"\n1. Интеративный метод\n2. Лямбда выражения\n3. LINQ- запрос\n Для выхода 0 или exit:");

switch (Console.ReadLine())

{

case "0":

case "exit":

return;

case "1":

IterDoubleFiltre(part);

break;

case "2":

LambdaDoubleFiltre(part);

break;

case "3":

LINQDoubleFiltre(part);

break;

}

}

}

catch{}

}

private void LINQDoubleFiltre(string part)

{

List<Film> films = getList();

List<Film> FiltredFilms = new List<Film>();

var selectedFilms =

from film in films

where film.genre.ToLower().Contains(part) && film.raiting%3==0 && film.raiting%7==0

select film;

FiltredFilms = selectedFilms.ToList();

PrintFiltredList(ref FiltredFilms, part);

}

private void LambdaDoubleFiltre(string part)

{

List<Film> films = getList();

List<Film> FiltredFilms = new List<Film>();

FiltredFilms = films.Where(film => film.genre.ToLower().Contains(part) && film.raiting%3==0 && film.raiting%7==0).ToList();

PrintFiltredList(ref FiltredFilms, part);

}

private void IterDoubleFiltre(string part)

{

List<Film> films = getList();

List<Film> FiltredFilms = new List<Film>();

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

{

if (films[i].genre.ToLower().Contains(part) && films[i].raiting%3==0 && films[i].raiting%7==0)

{

FiltredFilms.Add(films[i]);

}

}

PrintFiltredList(ref FiltredFilms, part);

}

private void LINQSort()

{

List<Film> films = getList();

IEnumerable<Film> sortedFilms =

from film in films

orderby film.title ascending, film.raiting descending

select film;

Console.WriteLine("{0,30} {1,10} {2,10} {3,3}", "Название фильма:", "Жанр:", "Премьера:", "Рейтинг:");

foreach (var film in sortedFilms)

{

Console.WriteLine("{0,30} {1,10} {2,10} {3,3}", film.title, film.genre, film.date.Date.ToString("d"), film.raiting);

}

}

private void LambdaSort()

{

List<Film> films = getList();

var sortedFilms=films.OrderBy(film=>film.title).ThenByDescending(film=>film.raiting);

Console.WriteLine("{0,30} {1,10} {2,10} {3,3}", "Название фильма:", "Жанр:", "Премьера:", "Рейтинг:");

foreach (var film in sortedFilms)

{

Console.WriteLine("{0,30} {1,10} {2,10} {3,3}", film.title, film.genre, film.date.Date.ToString("d"), film.raiting);

}

}

private void IterSort()

{

List<Film> films = getList();

for (int j = 0; j < films.Count; j++)

{

for (int i = 0; i < films.Count - j;)

{

if (i + 1 == films.Count) break;

if (films[i].title.CompareTo(films[i + 1].title) == 1)

{

Film temp = new Film(films[i + 1]);

films[i + 1] = films[i];

films[i] = temp;

i++;

}

else if (films[i].title.CompareTo(films[i + 1].title) == 0)

{

if (films[i].raiting.CompareTo(films[i].raiting) == -1)

{

Film temp = new Film(films[i + 1]);

films[i + 1] = films[i];

films[i] = temp;

i++;

}

else i++;

}

else i++;

}

}

/\*films.Sort(delegate(Film film1, Film film2)

{

if (film1.title.CompareTo(film2.title) == 0)

{

if (film1.raiting.CompareTo(film2.raiting) == -1)

return 1;

//else return -1;

}

else return film1.title.CompareTo(film2.title);

});\*/

Console.WriteLine("{0,30} {1,10} {2,10} {3,3}", "Название фильма:", "Жанр:", "Премьера:", "Рейтинг:");

foreach (var film in films)

{

Console.WriteLine("{0,30} {1,10} {2,10} {3,3}", film.title, film.genre, film.date.Date.ToString("d"), film.raiting);

}

}

private void LINQFiltreByDate(int startYear, int endYear)

{

List<Film> films = getList();

List<Film> FiltredFilms = new List<Film>();

var selectedFilm = from film in films

where film.date.Year < endYear&& film.date.Year>startYear

select film;

FiltredFilms = selectedFilm.ToList();

PrintFiltredList(ref FiltredFilms, startYear+"-"+endYear);

}

private void LambdaFiltreByDate(int startYear, int endYear)

{

List<Film> films = getList();

List<Film> FiltredFilms = new List<Film>();

FiltredFilms = films.Where(film => film.date.Year < endYear && film.date.Year>startYear).ToList();

PrintFiltredList(ref FiltredFilms, startYear + "-" + endYear);

}

private void IterFiltreByDate(int startYear, int endYear)

{

List<Film> films = getList();

List<Film> FiltredFilms = new List<Film>();

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

{

int filmYear = films[i].date.Year;

if (filmYear> startYear && filmYear<endYear)

{

FiltredFilms.Add(films[i]);

}

}

PrintFiltredList(ref FiltredFilms, startYear + "-" + endYear);

}

private void LINQFiltreByName(char key)

{

List<Film> films = getList();

List<Film> FiltredFilms = new List<Film>();

var selectedFilm= from film in films

where film.title[0] < key

select film;

FiltredFilms=selectedFilm.ToList();

PrintFiltredList(ref FiltredFilms, key.ToString());

}

private void LambdaFiltreByName(char key)

{

List<Film> films = getList();

List<Film> FiltredFilms = new List<Film>();

FiltredFilms = films.Where(film => film.title[0] < key).ToList();

PrintFiltredList(ref FiltredFilms, key.ToString());

}

private void IterFiltreByName(char key)

{

List<Film> films = getList();

List<Film> FiltredFilms = new List<Film>();

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

{

if (films[i].title[0] < key)

{

FiltredFilms.Add(films[i]);

}

}

PrintFiltredList(ref FiltredFilms, key.ToString());

}

private void LINQFiltreByPart(string part)

{

List<Film> films = getList();

List<Film> FiltredFilms = new List<Film>();

var selectedFilm = from film in films

where film.title.ToLower().Contains(part)

select film;

FiltredFilms = selectedFilm.ToList();

PrintFiltredList(ref FiltredFilms, part);

}

private void LambdaFiltreByPart(string part)

{

List<Film> films = getList();

List<Film> FiltredFilms = new List<Film>();

FiltredFilms = films.Where(film => film.title.Contains(part)).ToList();

PrintFiltredList(ref FiltredFilms, part);

}

private void IterFiltreByPart(string part)

{

List<Film> films = getList();

List<Film> FiltredFilms = new List<Film>();

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

{

if (films[i].title.Contains(part))

{

FiltredFilms.Add(films[i]);

}

}

PrintFiltredList(ref FiltredFilms, part);

}

private void PrintFiltredList(ref List<Film> filteredList, string key)

{

Console.Clear();

Console.WriteLine("{0,30} {1,10} {2,10} {3,3}", "Название фильма:", "Жанр:", "Премьера:", "Рейтинг:");

if (filteredList.Count == 0)

{

Console.WriteLine("По вашему запросу ничего не найдено.");

return;

}

Console.WriteLine("\nСписок отфильтрованных по {0}:", key);

foreach (var film in filteredList)

{

Console.WriteLine("{0,30} {1,10} {2,10} {3,3}", film.title, film.genre, film.date.Date.ToString("d"), film.raiting);

}

}

public static List<Film> getList()

{

List<Film> listFilms = new List<Film>();

listFilms.Add(new Film());

listFilms.Add(new Film("Пролетая над гнездом кукушки", "Драма", "19 11 1975", 89));

listFilms.Add(new Film("12 разгневанных мужчин", "Драма","10 4 1957", 82));

listFilms.Add(new Film("12 разгневанных мужчин", "Драма","10 4 1957", 63));

listFilms.Add(new Film("Молчание ягнят", "Триллер", "30 1 1991", 84));

listFilms.Add(new Film("Одержимость", "Драма", "16 1 2014", 84));

listFilms.Add(new Film("Побег из Шоушенка","Драма", "10 9 1994", 95));

listFilms.Add(new Film("Зёленая миля", "Фэнтази", "6 12 1999", 84));

listFilms.Add(new Film( "Список Шиндлера", "Биография","30 11 1993", 91));

listFilms.Add(new Film( "Пианист", "Биография","24 5 2002", 84));

listFilms.Add(new Film( "Пианист", "Биография","24 5 2002", 63));

listFilms.Add(new Film( "Американская история Х", "Драма","30 10 1998", 84));

listFilms.Add(new Film("Остров проклятых", "Триллер", "13 2 2010", 84));

return listFilms;

}

}

1. Класс ProgramException

internal class ProgramException : SystemException

{

public ProgramException(string message)

{

Console.WriteLine(message);

}

}

1. **Выводы:**

В ходе выполнения данной лабораторной работы мною были освоены методы работы со строками, списками и словарем; методы выборки. При этом была разработана программная реализация трех задач, предложенных вариантом задания. Решением первой задачи являлась часть приложения, считывающая с консоли строку и разделяющая ее, решение второй задачи состояло из реализаций нескольких дискретных подзадач, предполагающих выполнение различных операций, решение третьей задачи заключалось в работе со списками итеративным методом, с помощью лямбда выражений и LINQ-запросов.

Для написания программы были использованы стандартные методы и свойства стандартных классов, а также разработан ряд собственных классов. Реализация каждой отдельной подзадачи была заключена в особый класс, содержащий метод, производящий решение подзадачи и выводящий результат на консоль, а так же дополнительные методы.

Диалог пользователя с программой в консоли был реализован посредством главного меню, каждый пункт которого содержал подзадачу. После выполнения подзадачи пользователю предлагался диалог о повторе, позволяющий продолжить работу с данной подзадачей или вернуться в меню, в самом же меню так же осуществлен диалог для выхода из программы.

В ходе выполнения работы было рассмотрено большинство основных средств и конструкции языка C#.