Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
Рязанский государственный радиотехнический университет им. В.Ф. Уткина
Кафедра электронных вычислительных машин

К ЗАЩИТЕ
руководитель КП
_____ С.И. Елесина
«___»_____ 2020 г.

КУРСОВАЯ РАБОТА

по дисциплине

«Основы алгоритмизации и ООП»

на тему

«Разработка приложения
с использованием динамических структур данных»

Выполнил студент группы 940
Башев К.С.

_____
дата сдачи на проверку, подпись

Руководитель проекта
к.т.н., доцент Елесина С.И.

_____ _____
оценка дата защиты, подпись

Рязань 2020 г.

Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования
Рязанский государственный радиотехнический университет имени В.Ф. Уткина

Кафедра ЭВМ

ЗАДАНИЕ

на курсовую работу

по дисциплине «Основы алгоритмизации и ООП»

Студенту ***Башеву К.С., гр. 940***

1. Тема работы: **«**Разработка приложения с использованием динамических структур данных»

2. **Срок сдачи студентом законченной работы** *27 мая 2020 г.*

3. **Руководитель работы:** *Елесина Светлана Ивановна, к.т.н., доцент, доцент кафедры ЭВМ РГРТУ*

4. **Исходные данные к работе**

1.Операционная система Windows XP/7/8/10

2.Язык программирования C/C++

3.Среда программирования MS Visual Studio C++

4.Индивидуальное задание

5. **Содержание пояснительной записки**

Задание

Содержание

Введение

5. Постановка задачи

6. Разработка алгоритмов

7. Разработка программы

8. Экспериментальная проверка программы

9. Руководство оператора.

Заключение.

Список использованных источников.

Приложение – листинги программ.

Задание выдано «14» марта 2020 г._____/ доцент каф. ЭВМ С.И. Елесина

Индивидуальное задание к курсовой работе по дисциплине «Основы алгоритмизации и ООП»

студенту группы 940 Башеву К.С.

Общие требования

Вариант задания связан с разработкой таблицы данных с использованием линейных однонаправленных списков.

Разрабатываемая программа должна обязательно выполнять следующие запросы:

- заполнение пустой таблицы;
- сохранение таблицы в файле;
- чтение таблицы из файла;
- вывод таблицы на экран;
- добавление элементов в таблицу;
- удаление элементов из таблицы;
- редактирование элементов таблицы;
- а также все запросы, которые указаны в индивидуальном задании.

Вызовы запросов должны осуществляться через систему меню с использованием средств визуального программирования *Visual Studio (Visual C++)*. Необходимо предусмотреть контроль ошибок пользователя при вводе данных. Результаты некоторых запросов должны выводиться в виде графиков или диаграмм.

При запуске приложение должно выдавать заставку, которая отражает назначение приложения.

Все элементарные действия должны быть оформлены в виде подпрограмм, а некоторые объявления и подпрограммы должны быть оформлены в виде модуля (модулей).

Индивидуальное задание

Вариант № 04. Домашняя кинотека

Информация о фильмах представлена в следующем виде:

- название;
- жанр;
- страна;
- год производства;
- режиссер;
- качество (формат);
- звук (оригинальный, дублированный, …);
- время (продолжительность).

Написать программу, которая выполняет следующие запросы:

- по названию выводится информация о фильме;
- вывод фильмов определенного жанра и страны;
- в алфавитном порядке выводит список фильмов определенного режиссера;
- определение и вывод процентного соотношения фильмов в зависимости от страны производителя (диаграмма).

Задание выдано «14» марта 2020 г._____/ доцент каф. ЭВМ С.И. Елесина

Оглавление

Введение

Целью курсового проектирования являются разработка и отладка приложения с использованием динамических структур данных, написанного на языке программирования C++ с использованием средств визуального программирования, предоставляемых средой разработки Microsoft Visual Studio.

Язык программирования C++ — компилируемый, статически типизируемый язык общего назначения, широко используемый для разработки различного программного обеспечения. На сегодняшний день это один из самых известных и популярных языков программирования.

Являясь языком высокого уровня и будучи пригодным для прикладного программирования, C++ позволяет вручную работать с памятью, что обеспечивает возможность разрабатывать действительно высокопроизводительные и быстрые GUI приложения. Это выгодно отличает C++ от других языков. Стандартная библиотека C++, а также различные пакеты инструментов для разработки desktop приложений делают его отличным выбором, для выполнения поставленной в данной работе задачи.

Кроме того, возможность работы с памятью делает этот язык идеальным для системного программирования – изначально язык Си как раз-таки и разрабатывался для разработки операционной системы UNIX. На нём в значительной степени написаны самые известные на сегодняшний день ОС: Microsoft Windows, Linux и Mac OS.

Так же благодаря высокой производительности этот язык нашёл своё применение в анализе изображений и различных графических элементах компьютерных игр со сложной и детализированной графикой.

C++ является компилируемым языком, а это значит, что компилятор транслирует исходный код на C++ в исполняемый файл, который содержит набор машинных инструкций. Но разные платформы имеют свои особенности, поэтому скомпилированные программы нельзя просто перенести с одной платформы на другую и там уже запустить. Однако на уровне исходного кода программы на C++ по большей степени обладают переносимостью, если не используются какие-то специфичные для текущей ос функции. А наличие компиляторов, библиотек и инструментов разработки почти под все распространенные платформы позволяет компилировать один и тот же исходный код на C++ в приложения под эти платформы.

В отличие от Си язык C++ позволяет писать приложения в объектно-ориентированном стиле, представляя программу как совокупность взаимодействующих между собой классов и объектов. Что упрощает создание крупных приложений.

Динамическая структура данных – это любая структура данных, объём которой не является фиксированным. В данном проекте используется структура «линейный односвязный список», каждый элемент которой посредством указателя связан со следующим элементом.

Использование динамических структур данных позволяет хранить любые объёмы данных, что, разумеется, выгодно конечному пользователю. Возможность же работать с памятью, которую даёт C++, позволяет использовать их с наибольшей эффективностью.

1. Постановка задачи

1.1. Выбор исходных данных, их типов и структур, а также ограничений на них

Выделим исходные данные для описания фильмов.

- Название – тип строка
- Жанр – тип строка
- Страна производства – тип строка
- Год производства – тип строка
- Режиссёр – тип строка
- Качество – тип строка
- Озвучка – тип строка
- Длительность – тип строка

Создадим структуру Movie, содержащую эти данные:

```
struct Movie
{
	string name; // Название
	string genre; // Жанр
	string country; // Страна
	string productionYear; // Год производства
	string producer; // Режиссёр
	string format; // Качество
	string sound; // Звук
	string time; // Продолжительность

	Movie* link; // Указатель на следующий элемент списка
};
```

1.2. Детализация функций программы

Программа включает в себя следующие функции:

- Ввод таблицы с клавиатуры
- Загрузка таблицы из файла
- Вывод таблицы на экран
- Сохранение таблицы в файл
- Дополнение таблицы
- Редактирование таблицы
- Удаление элемента таблицы
- Вывод информации о фильме по введённому названию
- Вывод списка фильмов определённого жанра и страны
- Вывод списка фильмов определённого режиссёра в алфавитном порядке
- Вывод диаграммы процентного соотношения стран-производителей фильмов
- Вывод справки
- Выход из программы

1.3. Разработка интерфейса программы для ввода исходных данных и вывода результатов работы программы

Интерфейс программы включает в себя следующие компоненты:

- Главная форма – содержит кнопки, посредством нажатия которых осуществляется доступ к функционалу программы, панели, на которых располагаются элементы, обеспечивающие этот функционал и компонент dataGridView, используемый для вывода таблицы на экран. Для различных запросов пользователя доступны разные компоненты программы – некоторые из них могут опционально скрываться или быть недоступны, для взаимодействия.
- Диалоговое окно сохранения файла – осуществляет запись данных в указанный пользователем файл.
- Диалоговое окно открытия файла – осуществляет чтение данных из указанного пользователем файла.
- Панель ввода элемента – вызывается при запросе пользователем ввода таблицы с клавиатуры, редактирования или дополнения таблицы, а также при запросе удаления элемента таблицы. Содержит текстовые поля для ввода данных и элемент listBox, в котором отображаются текущие данные, что позволяет выбрать конкретный элемент для редактирования или удаления.

  На панели присутствуют кнопки «продолжить» и «назад», которые изменяют данные таблицы при правильно введённых данных, или скрывают панель при нажатии на них.
- Панель поиска элемента - аналогично вызывается при нажатии кнопок с запросами по индивидуальному заданию. Присутствуют текстовые поля для ввода информации, кнопка «ввод» - выполняющая запрос при правильно введённой информации и кнопка «назад» - скрывающая панель.
- Диаграмма – элемент Chart, отображающий соотношение стран-производителей фильмов в кинотеке. Представляет собой круговую диаграмму.
- Форма заставки

2. Разработка алгоритма

2.1. Выделение основных частей приложения

Для упрощения создания приложение было разделено на 3 модуля:

- Класс MovieLibrary, для работы со списком как с простейшей базой данных – осуществляет основные манипуляции с динамическим списком.
- Класс MyForm, для работы с графическим интерфейсом, обеспечивает логику программы, обмен данными с пользователем и их фильтрацию.
- Класс DataCounter, для составления выборки из стран-производителей и количества фильмов, произведённых в каждой из них. Предназначен для анализа кинотеки и передачи готового результата по запросу на диаграмму.

2.2. Схемы алгоритмов.

Далее представлены алгоритмы, используемые в программе представлены ниже (см. Рисунок 1 - 10).

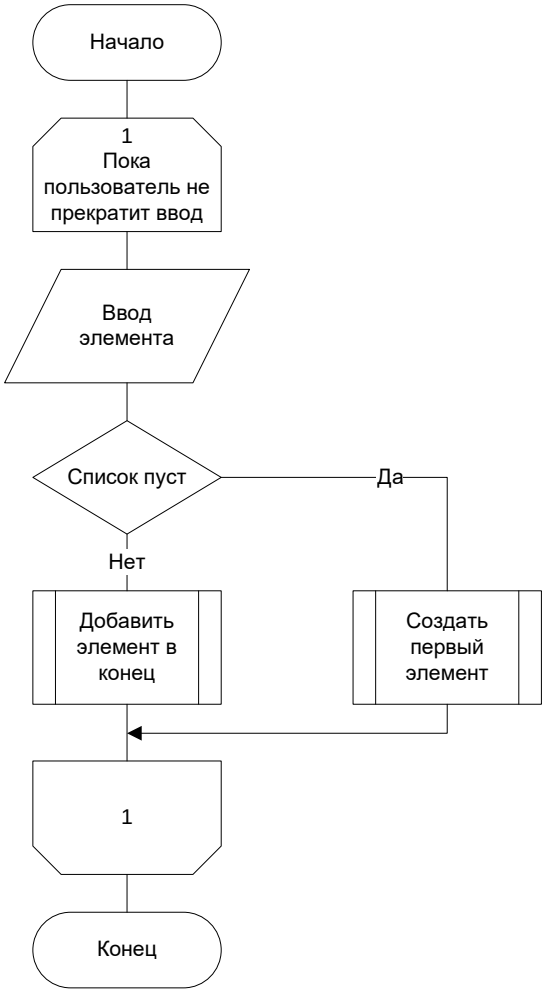

Рисунок 1— Заполнение таблицы с клавиатуры

Рисунок 2 — Вывод таблицы на экран

Рисунок 3 — Сохранение таблицы в файл

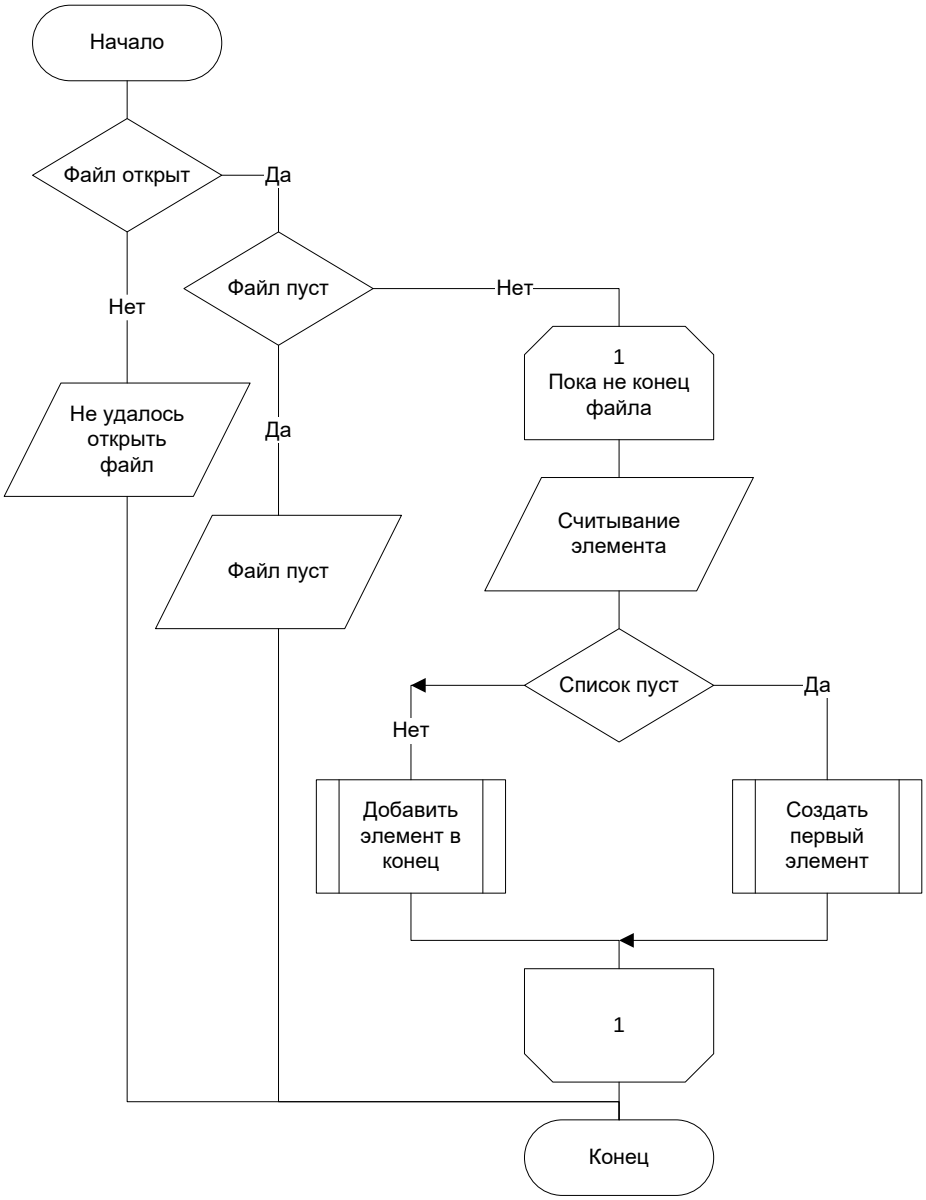

Рисунок 4 — Загрузка таблицы из файла

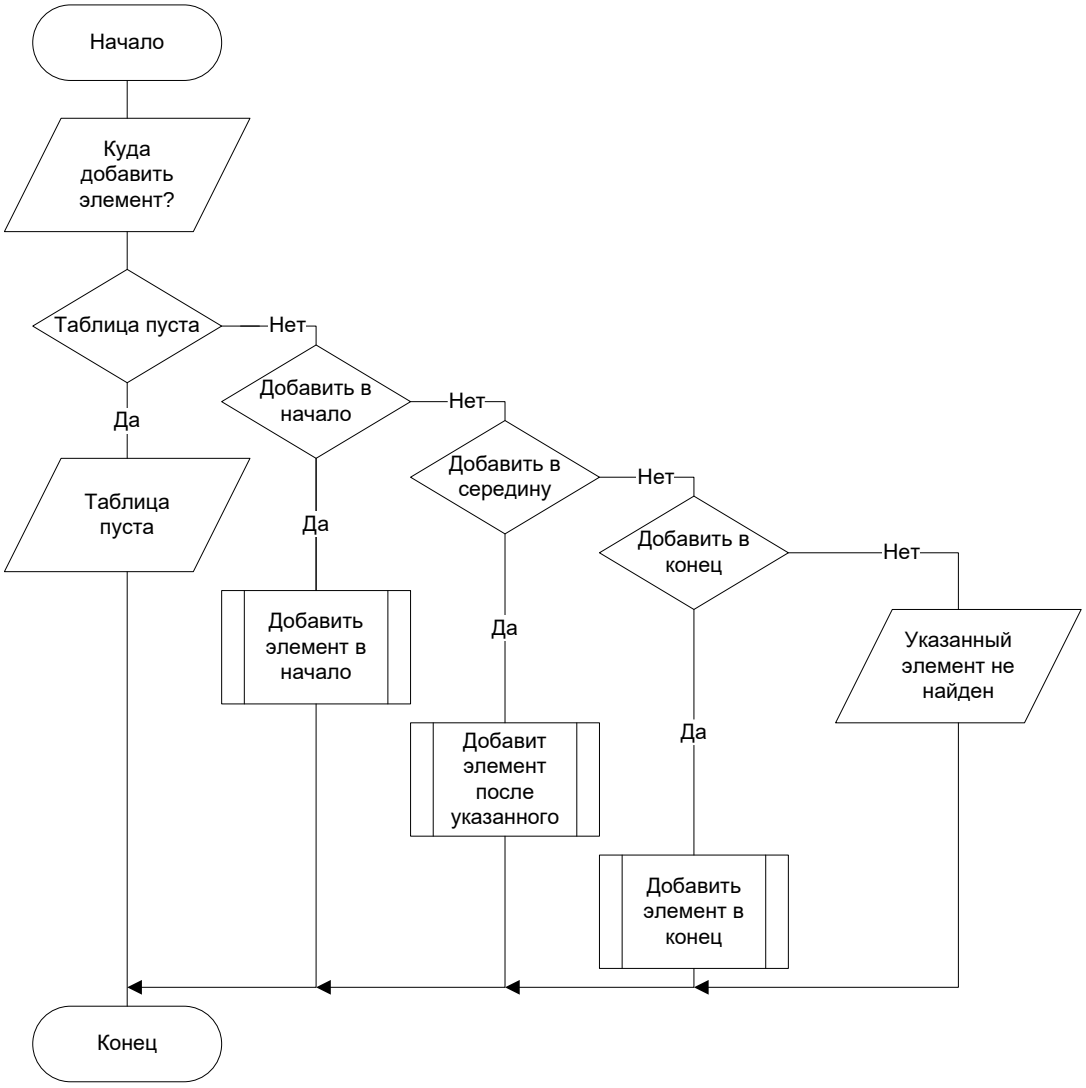

Рисунок 5 — Дополнить таблицу

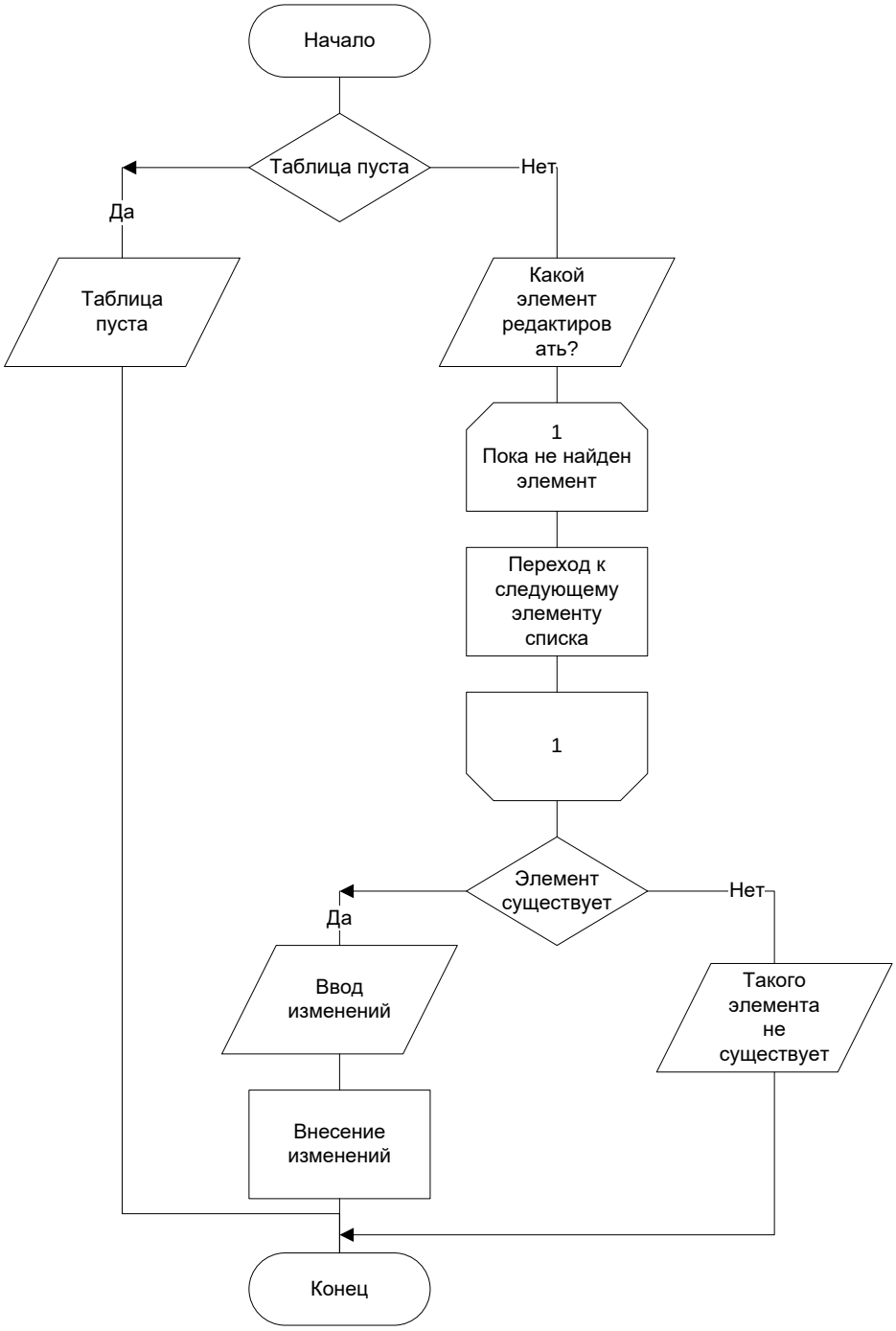

Рисунок 6 — Редактировать таблицу

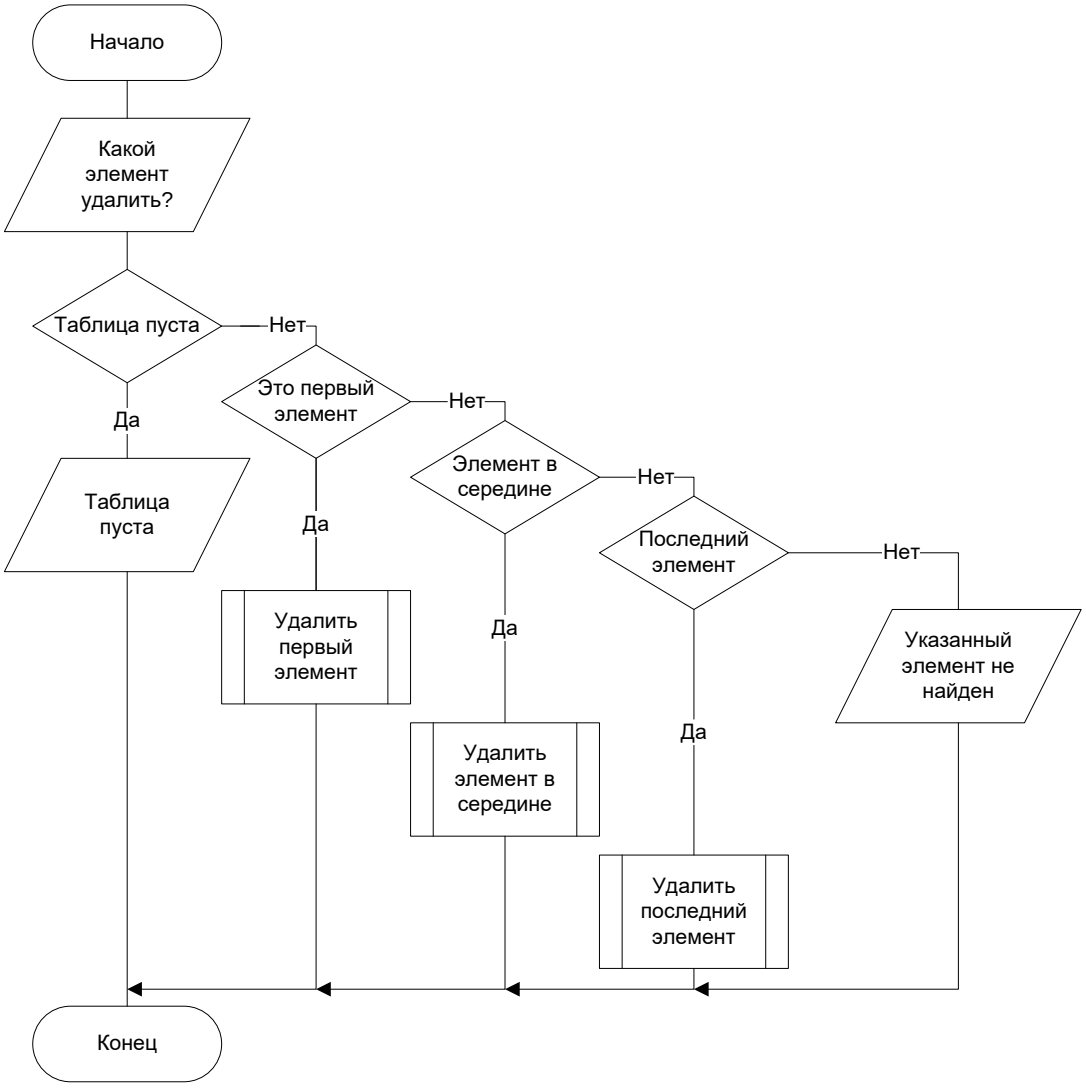

Рисунок 7 — Удаление элемента из таблицы

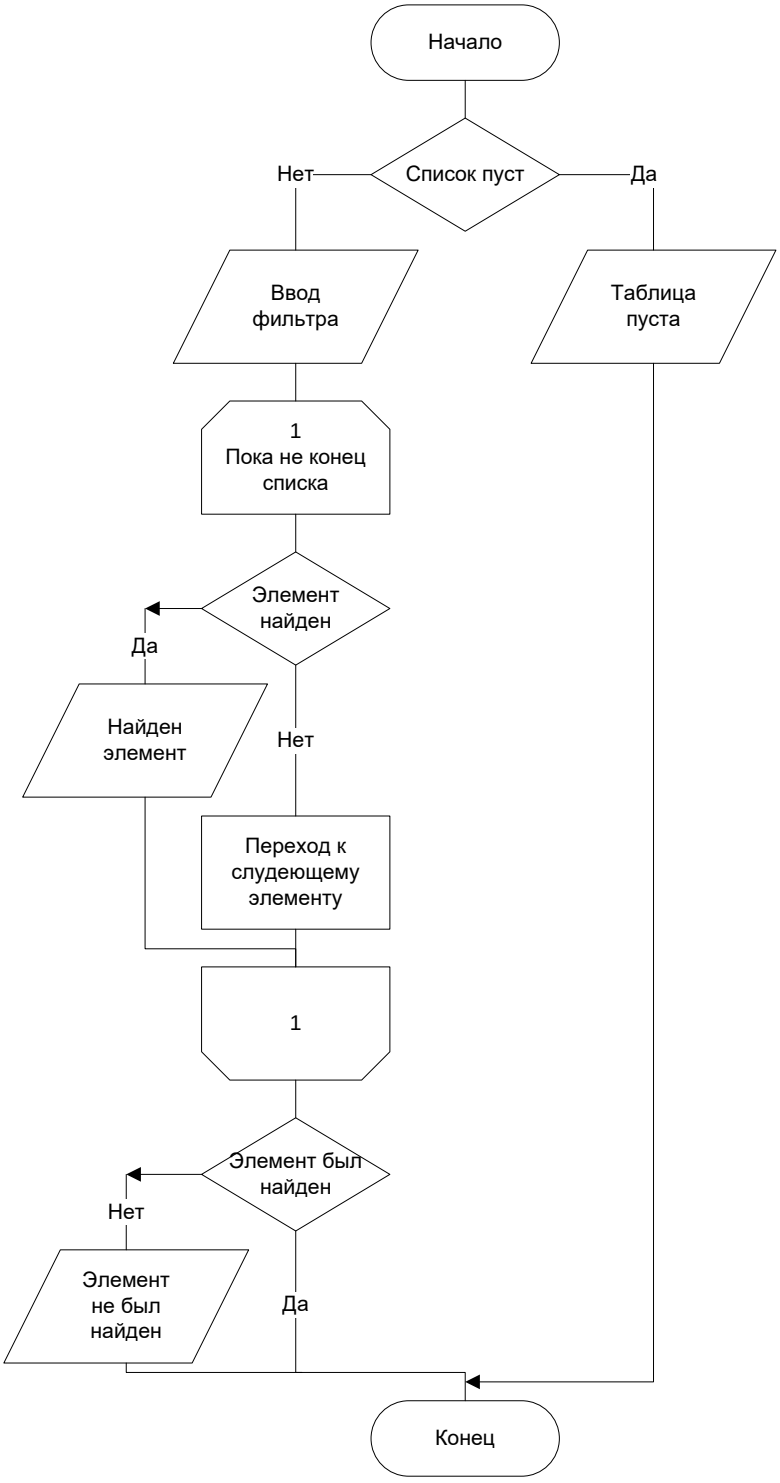

Рисунок 8 — Поиск по списку

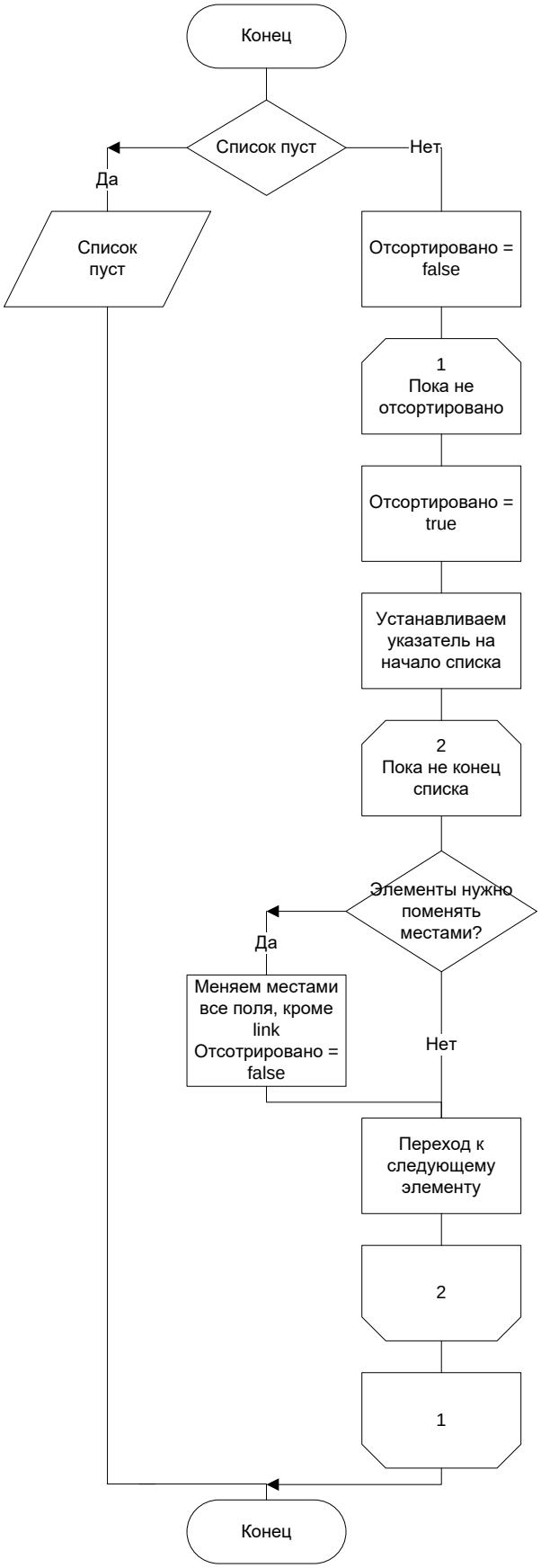

Рисунок 9 — Сортировка списка

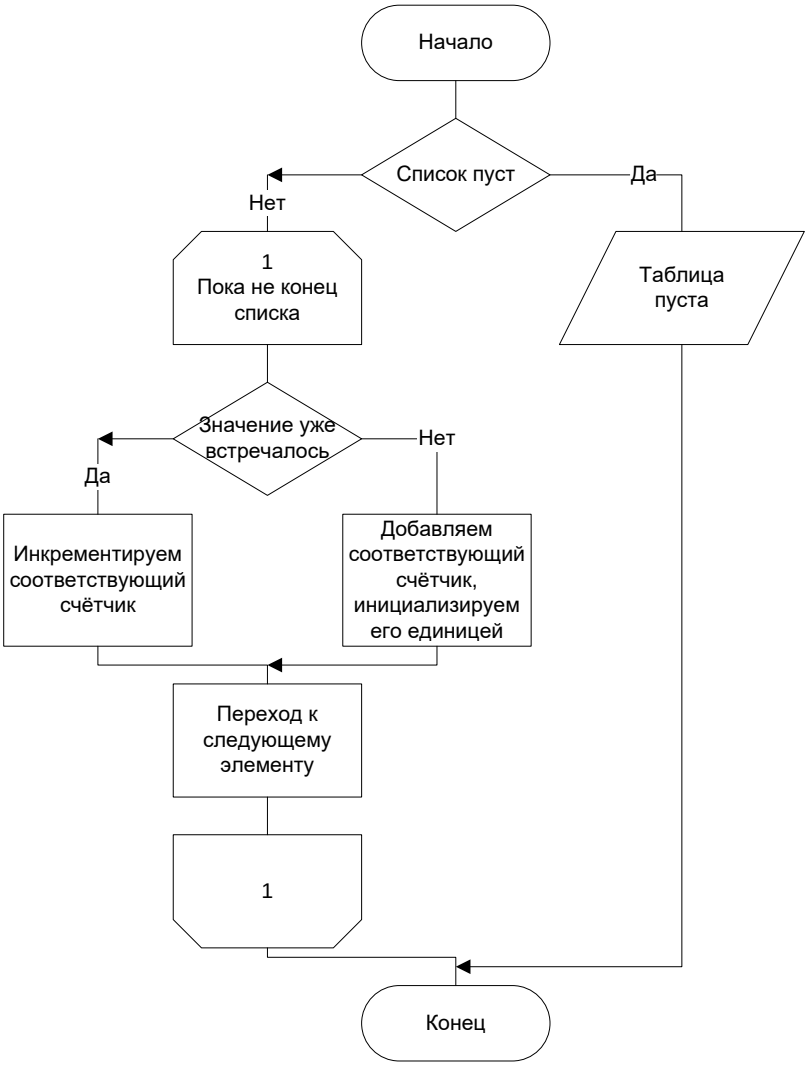

Рисунок 10 — Анализ данных для заполнения диаграммы

3. Разработка программы

3.1. Выбор языка программирования

Для написания программы по составленным алгоритмам выберем среду разработки Microsoft Visual Studio C++ и платформу .Net, поскольку она обеспечивает возможность разработки визуального интерфейса, понятного пользователю.

Таким образом, язык программирования – C++/CLI.

3.2. Описания исходных текстов программ и структур входных, выходных и промежуточных данных

Как уже упоминалось ранее, программа состоит из 3 основных классов:

1. MovieLibrary – для базовых манипуляций со списком
2. MyForm – для реализации интерфейса
3. dataCounter – для подготовки данных к отображению на диаграмме.

Рассмотрим подробнее каждый из них.

Класс MovieLibrary содержит в себе указатель на начало списка, путь к файлу с данными и флаг состояния – пуст ли список. Код этого класса частично приведён в приложениях А и Б.

Этот класс так же содержит методы для работы с односвязным списком:

- Базовые методы для доступа к списку (получить указатель на элемент под определённым номером – код метода getElAt представлен в приложении, присвоить значение элементу с указанным номером, получить указатель на первый элемент и узнать, пуст ли список.)
- Методы взаимодействия с файлом: saveList для записи в файл и loadList для чтения из файла соответственно – возвращают код, сообщающий о результате операции. Код этих методов представлен в приложении. Так же существует метод для изменения имени файла по умолчанию.
- Методы инициализации и дополнения списка: создать первый элемент, добавить элемент в начало, а также перегруженный метод метод addFilm для добавления фильмов в конец списка или после элемента с определённым номером – код второго варианта метода addFilm представлен в приложении.
- Группа методов для удаления элементов из таблицы: deleteFilm, deleteLastFilm и deleteTable для удаления последнего элемента, элемента с указанным номером и для удаления всего списка целиком соответственно. Код метода deleteFilm представлен в приложении.
- Метод для сортировки списка sort - представлен в приложении. Так же «в помощь» ему существует скрытый метод swap, меняющий два любых элемента списка местами. Все остальные методы открыты.

Класс MyForm содержит поля для сохранения результатов действий пользователя – поля selectedIndex, куда записывается номер выбранного

пользователем элемента и act, где хранится информация о последнем запросе пользователя. Так же присутствуют поля ML – где содержится адрес объекта класса MovieLibrary (этот адрес передаётся при инициализации формы через конструктор) - а так же head и current, указатели на начало кинотеки и на текущий элемент соответственно. Код этого класса частично приведён в приложениях Д и Е.

Рассмотрим наиболее важные методы класса MyForm:

- Методы showList и updateDiag соответственно отвечают за вывод таблицы на экран и за обновление данных диаграммы. Для отображения таблицы используется компонент dataGridView, для диаграммы – компонент Chart.
- Обработчики событий нажатия на кнопки «Ввести таблицу с клавиатуры», «Дополнить таблицу», «Удаление элементов» и «Редактирование элементов» открывают специальную панель (inputListPanel) для ввода данных пользователем. На ней расположены текстовые поля и кнопки ввода и отмены. Именно обработчик нажатия кнопки ввода играет ключевую роль в программе. Один из этих методов – addToList_Click (обработчик нажатия кнопки «Дополнить таблицу») – представлен в приложении.
- Обработчик кнопки ввода – inputEl_Click в зависимости от выбранного пользователем ранее действия (заполнение, редактирование, дополнение таблицы или удаление элементов из неё) вызывает соответствующий метод класса MovieLibrary, а также, следит за правильностью введённой пользователем информации. Код данного метода представлен в приложении.

  Для получения данных от пользователя используются компоненты TextBox и ListBox.
- Обработчик кнопки «назад» этой панели обновляет все данные, если это требуется и отображает или скрывает элементы управления в зависимости от состояния таблицы. Код метода stopInputList_Click представлен в приложении.
- Обработчики specialRequest1_Click и specialRequest2_Click и specialRequest_3Click работают схожим образом с описанными выше обработчиками кнопок изменения таблицы – отображают на экране панель (specialRequestsPanel) и передают управление на другой элемент (будет ожидаться нажатие кнопки вывод или назад).
- Обработчик нажатия кнопки вывод на упомянутой панели в зависимости от запроса пользователя выводит на экран таблицу с разными фильтрами (по названию или по жанру и стране) или формирует новый список, добавляя в него только соответствующие условию элементы, сортирует его и выводит в таблицу – так он работает с запросом на вывод фильмов конкретного режиссёра в алфавитном порядке. Код метода confirmButton_Click представлен в приложении.

- Обработчик нажатия кнопки cancelButton на этой панели работает аналогично кнопке «назад», описанной ранее.
- Обработчик specialRequest4_Click выводит на экран диаграмму или прячет её, если она уже выведена. Данные при этом обновляются.
- Так же в программе присутствует много вспомогательных элементов, вроде обработчиков нажатия клавиши в текстовом поле или изменения выделения в listBox. Один такой метод – fName_KeyPress приведён в приложении.

Класс dataCounter принимает указатель на объект класса MovieLibrary и на основе полученных данных в методе analyse заполняет два динамических массива std::vector – первый заполняется элементами типа string, содержащими уникальные названия стран (проверка уникальности осуществляется в методе thereIsSameStat), а второй – счётчиками, которые хранят количество упоминаний каждой страны. Код этих двух методов представлен в приложении Г. Код заголовочного файла этого класса со всеми функциями приведён в приложении В.

4. Экспериментальная проверка программы

Результаты работы программы представлены ниже (см. Рисунок 11 - 28)

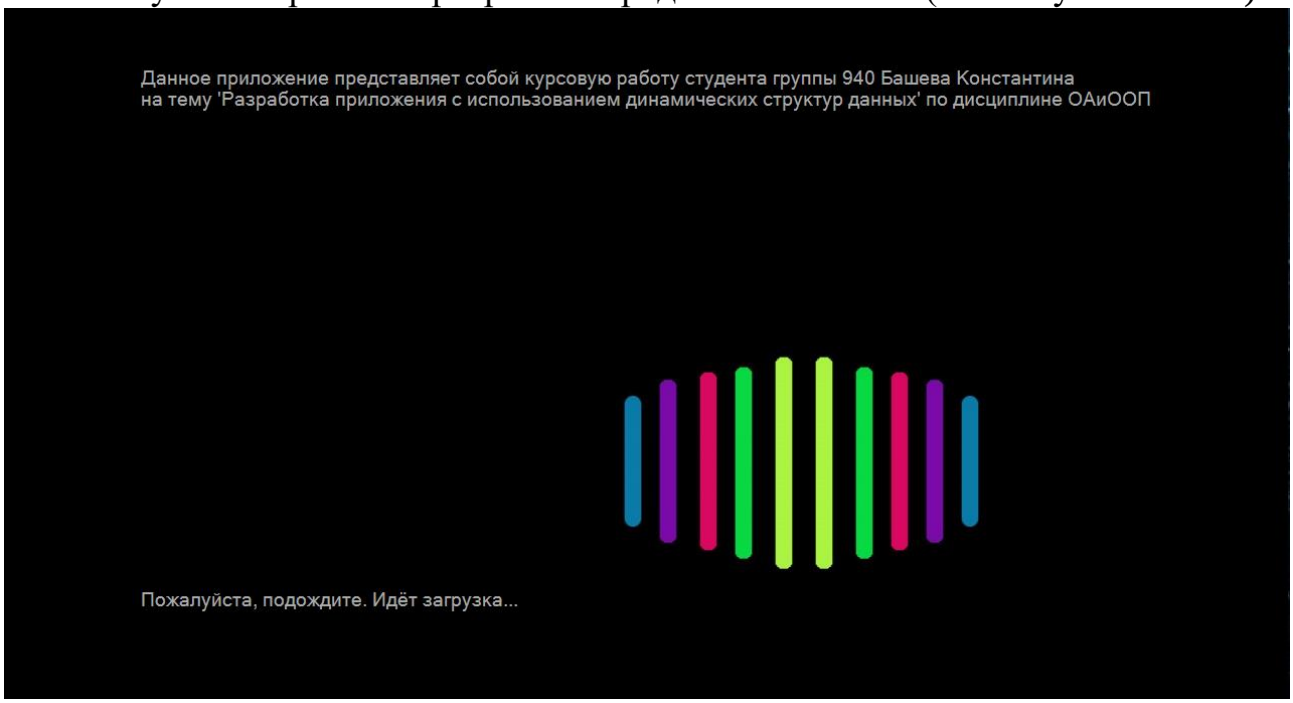

Рисунок 11 — Заставка

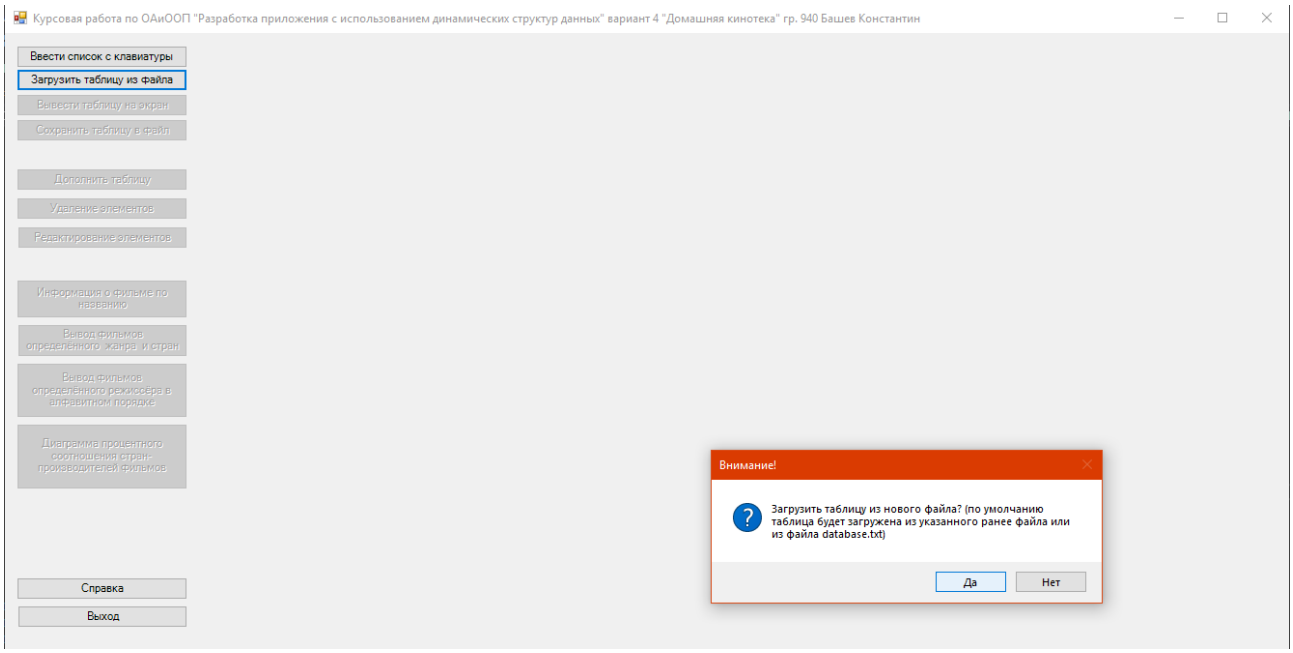

Рисунок 12 — Открыть файл

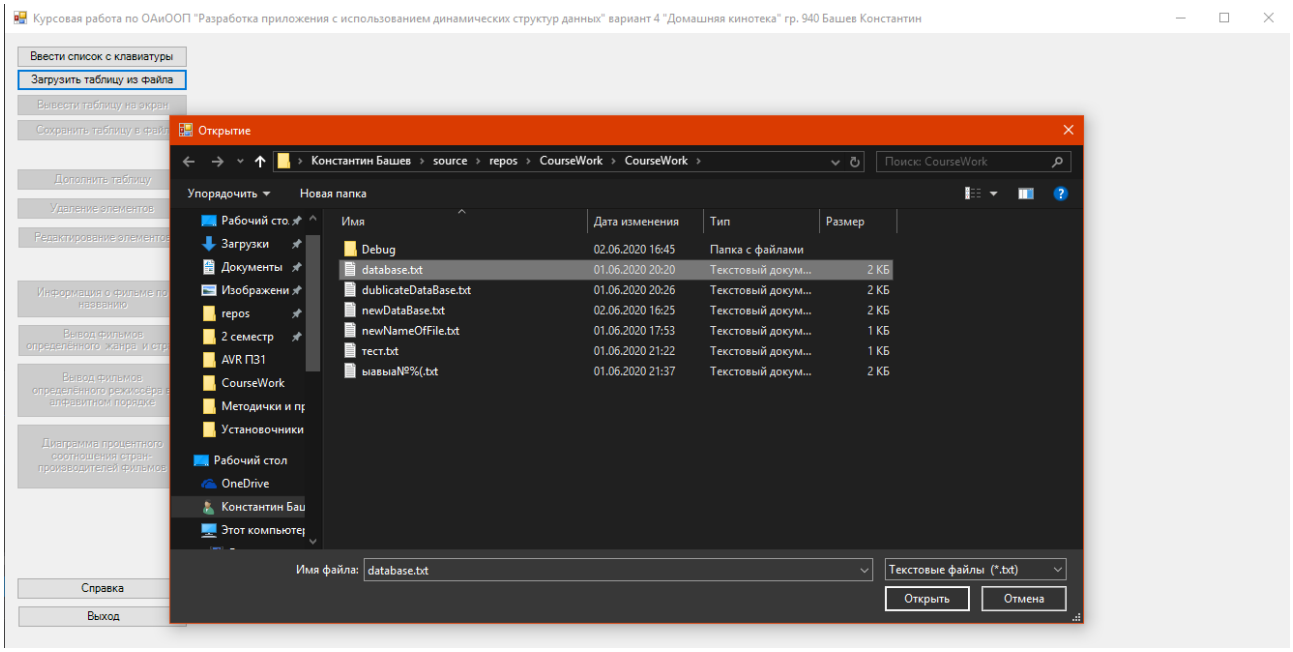

Рисунок 13 — Диалог открытия файла

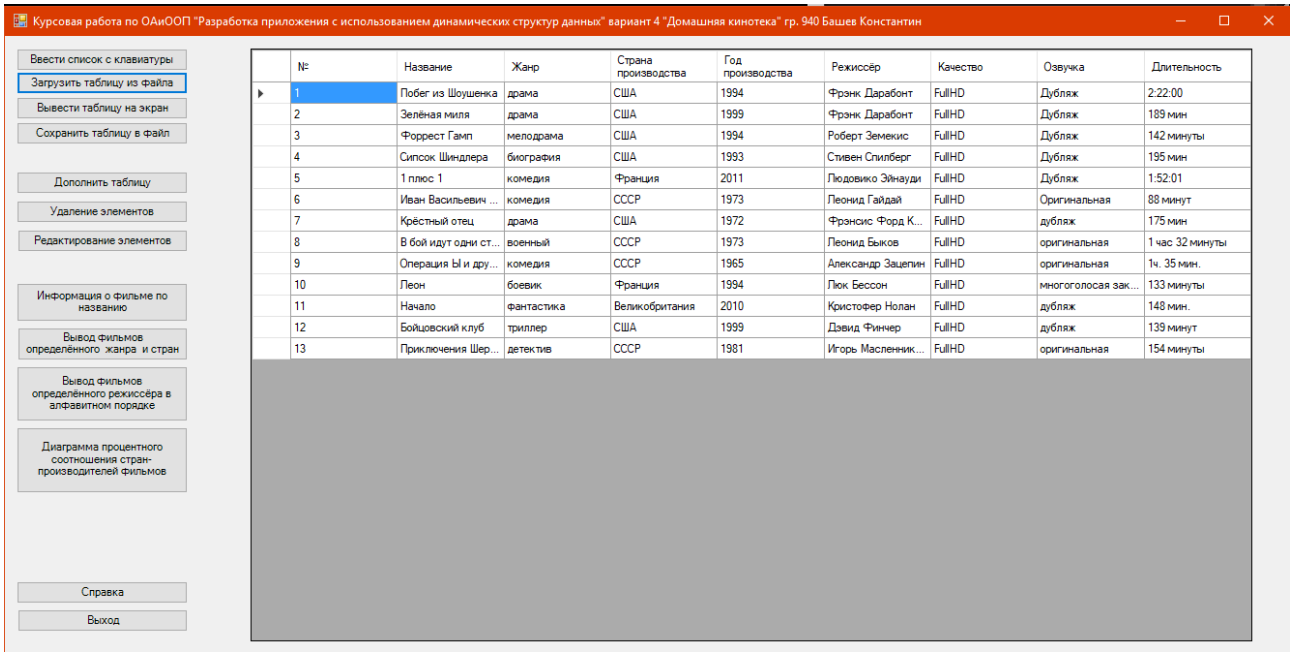

Рисунок 14 — Вывели таблицу на экран

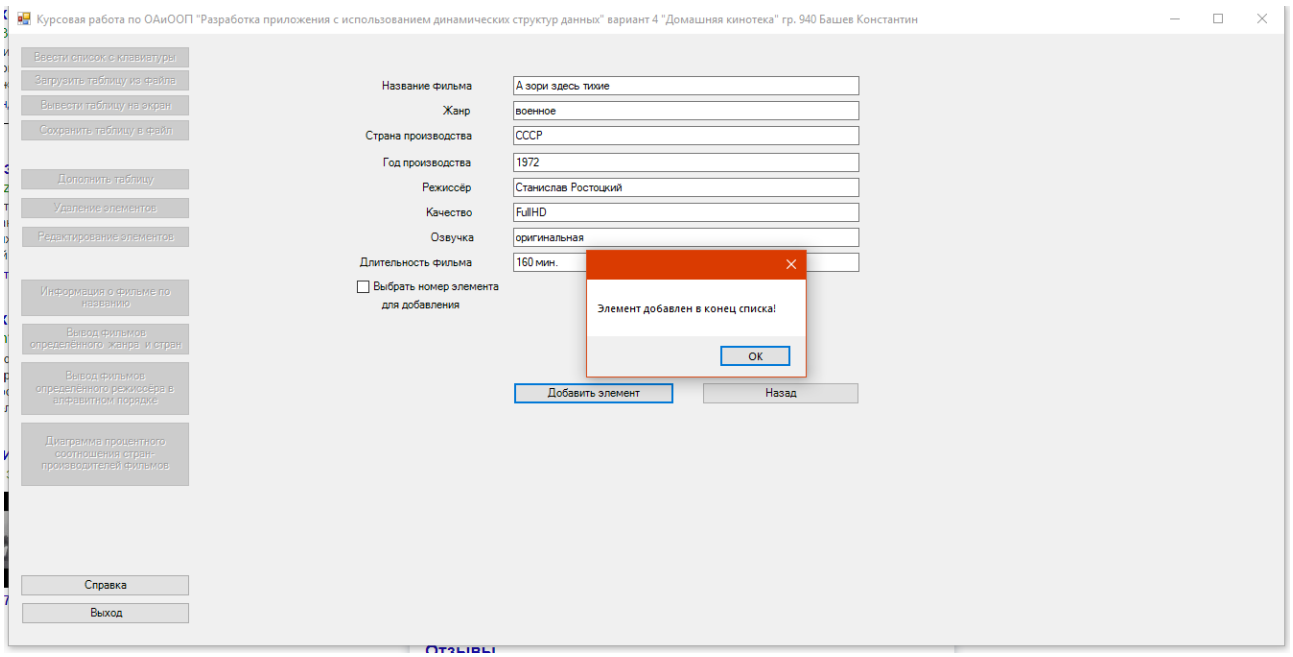

Рисунок 15 — Добавили элемент в конец таблицы

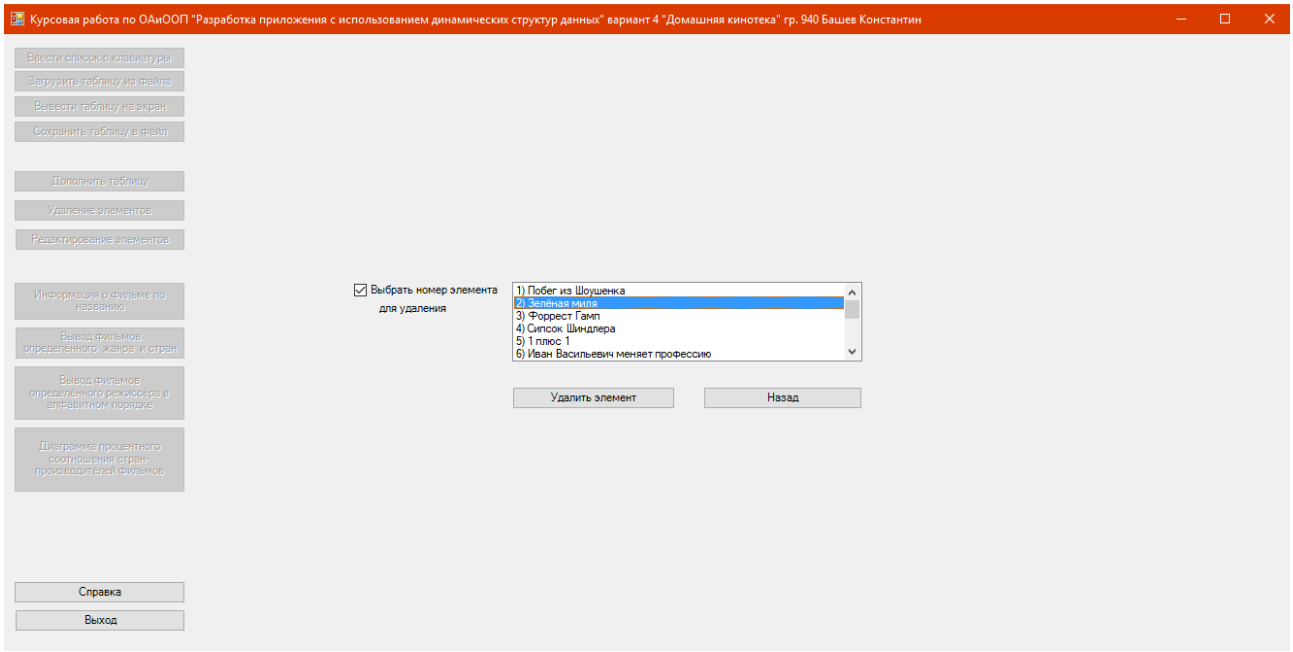

Рисунок 16 — Удалили второй элемент таблицы

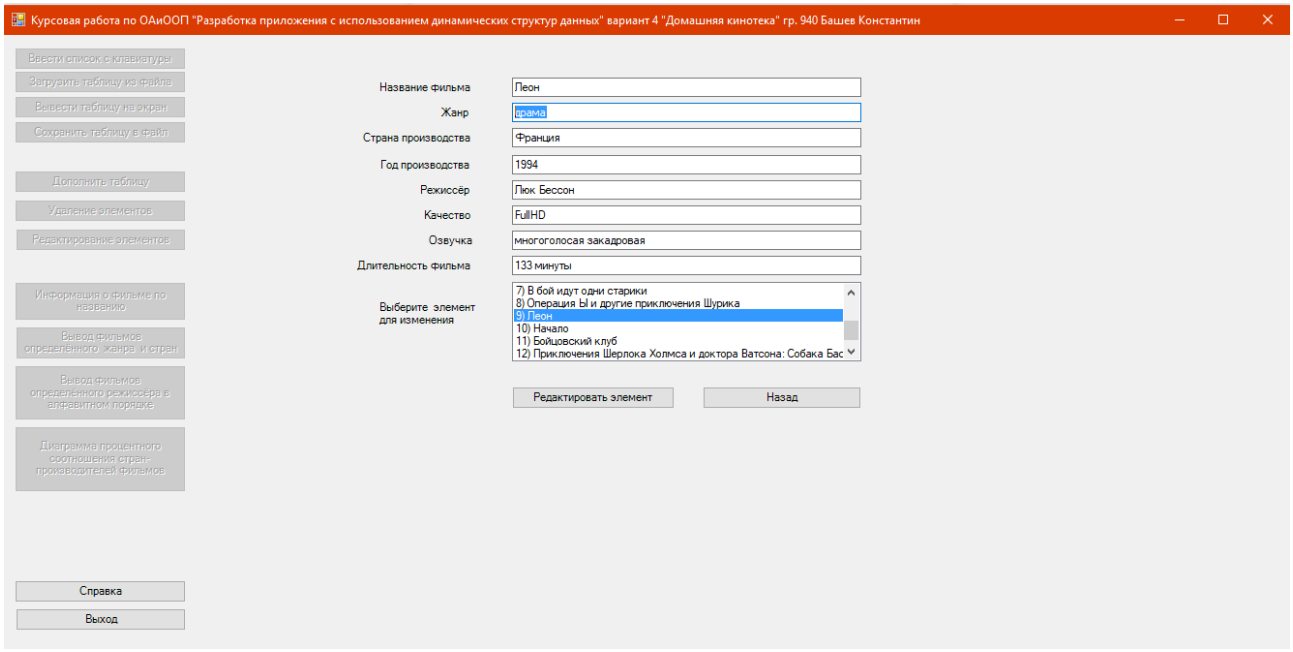

Рисунок 17 — Отредактировали фильм "Леон"

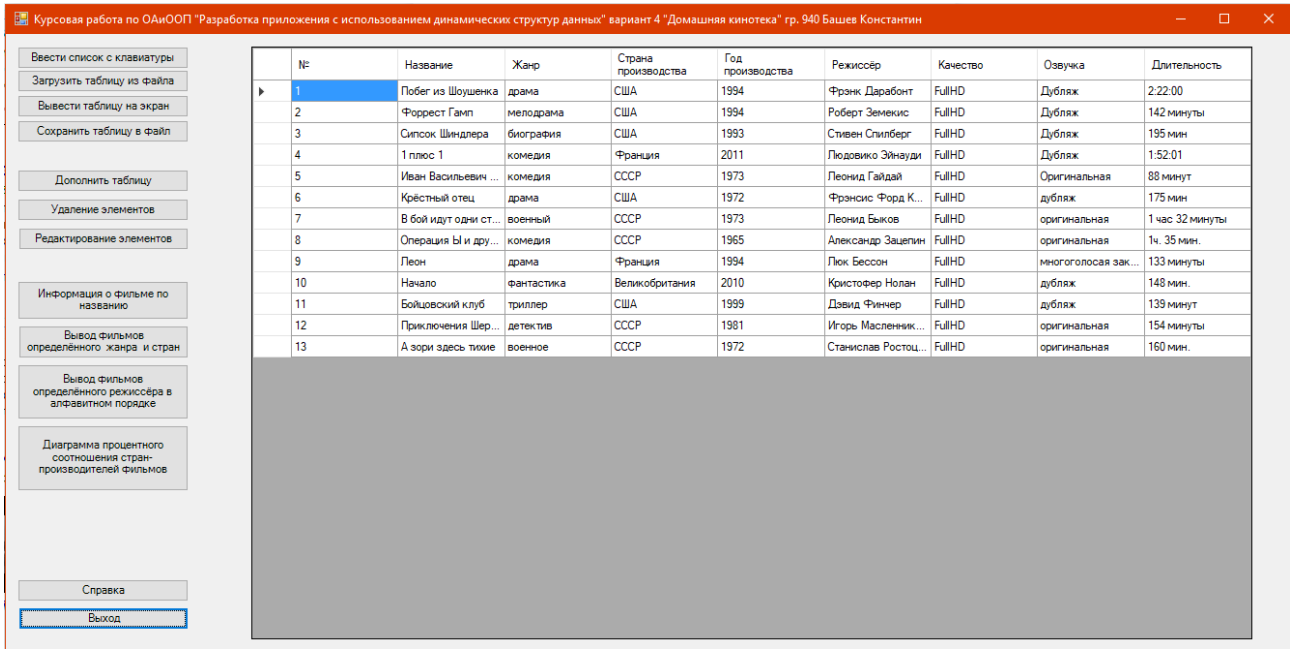

Рисунок 18 — Изменённая таблица

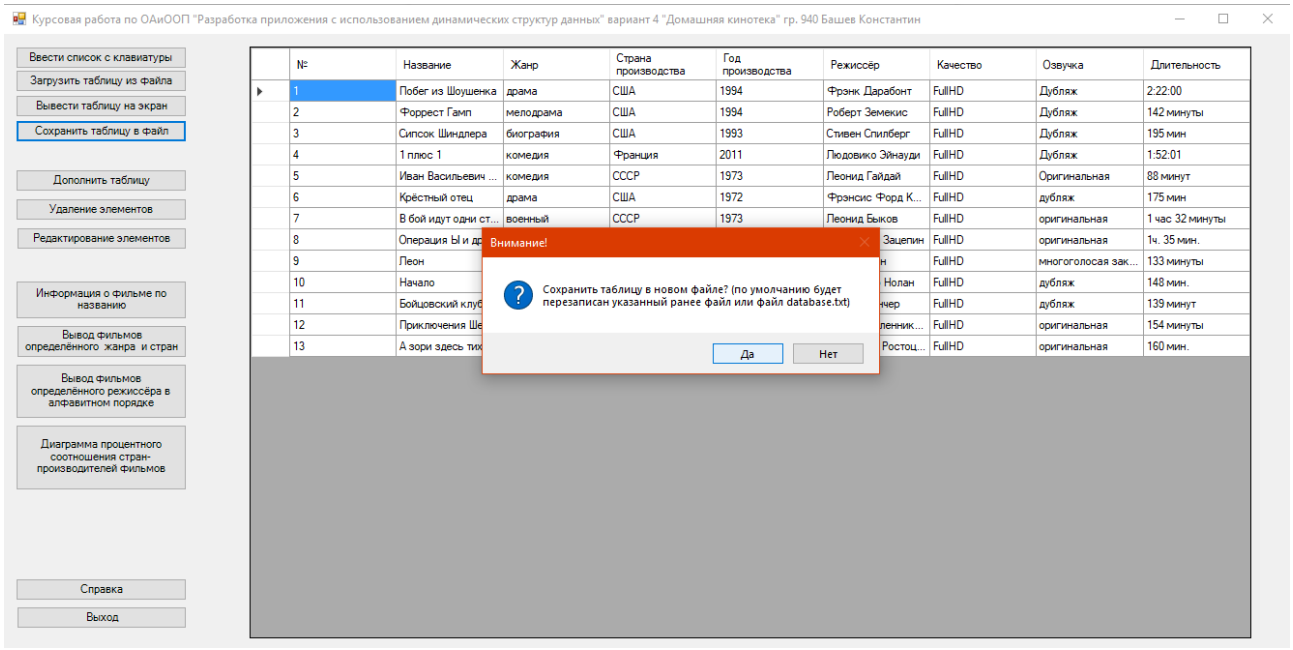

Рисунок 19 — Сохранить таблицу

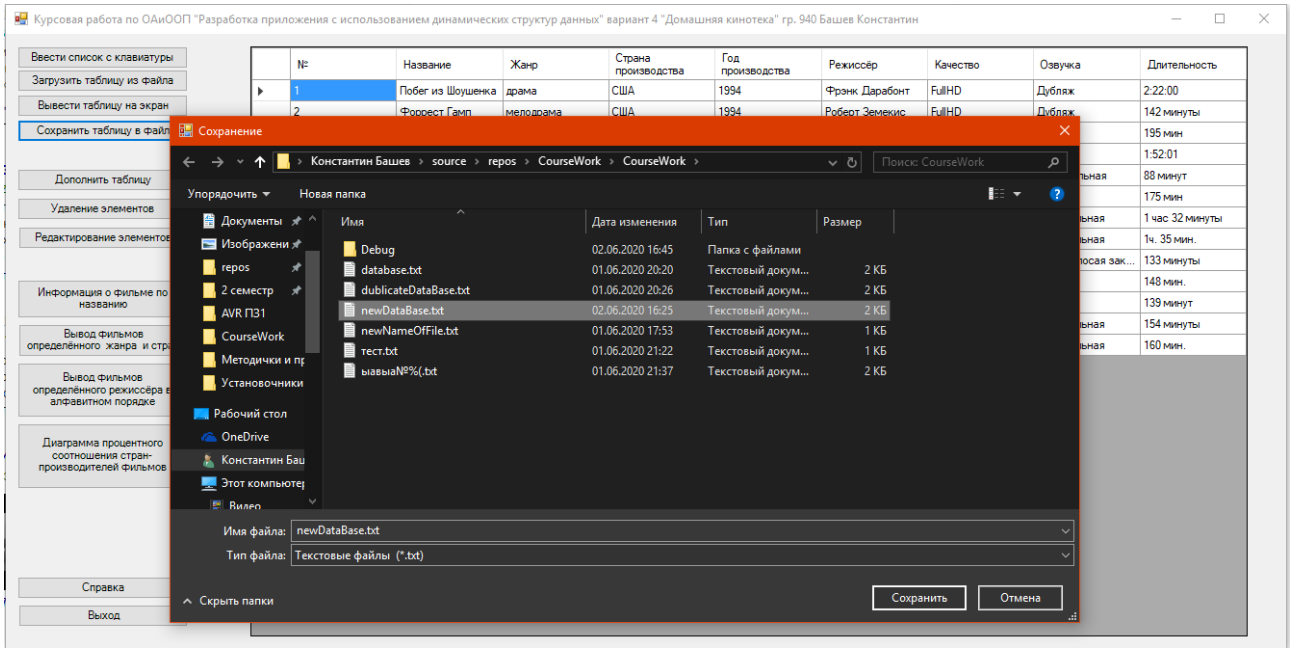

Рисунок 20 — Диалог сохранения в файле

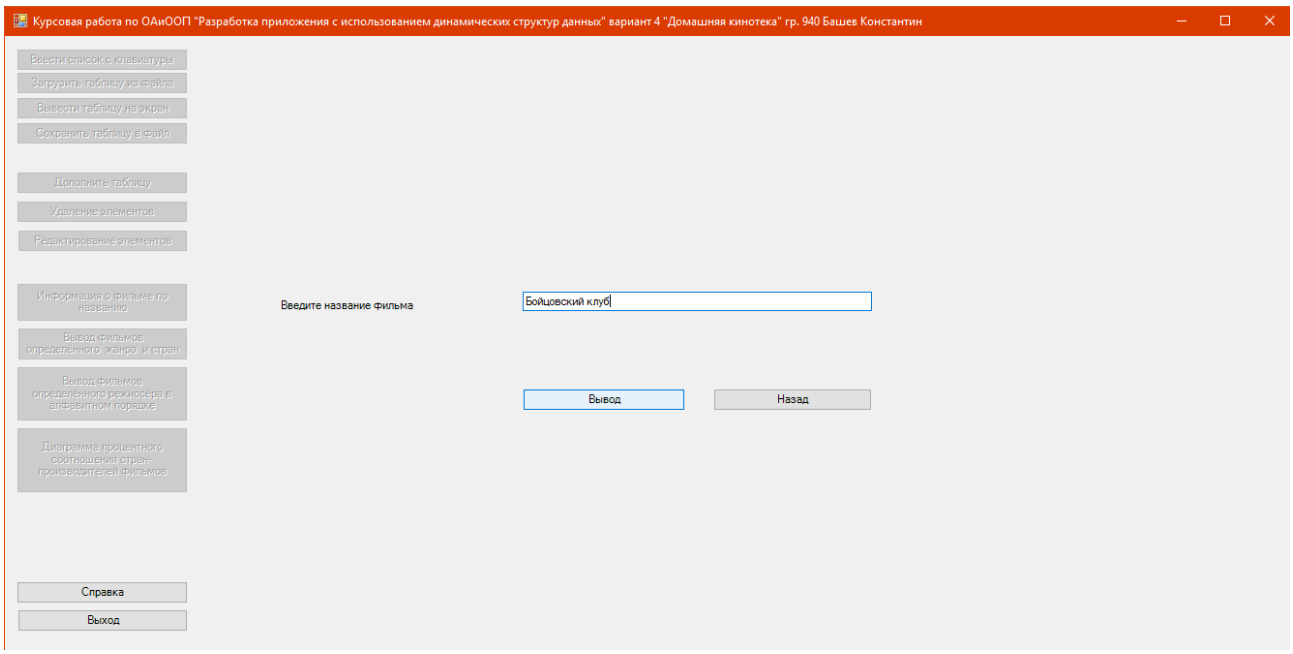

Рисунок 21 — Поиск по названию

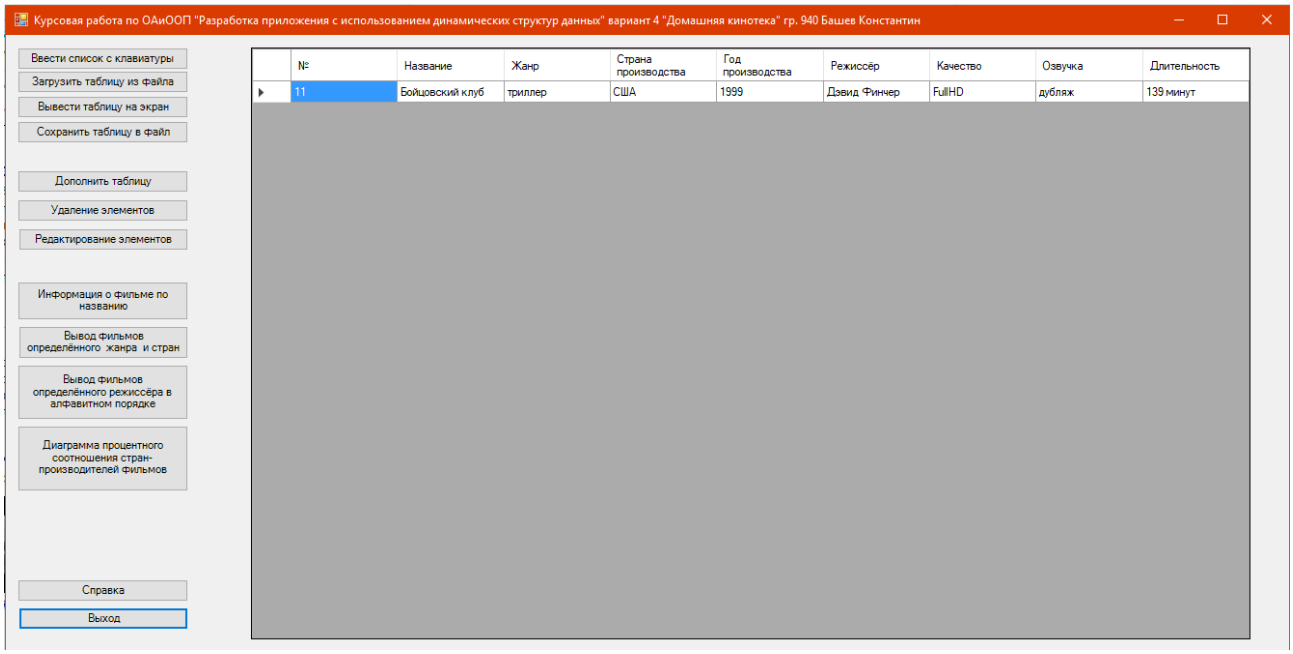

Рисунок 22 — Результат поиска

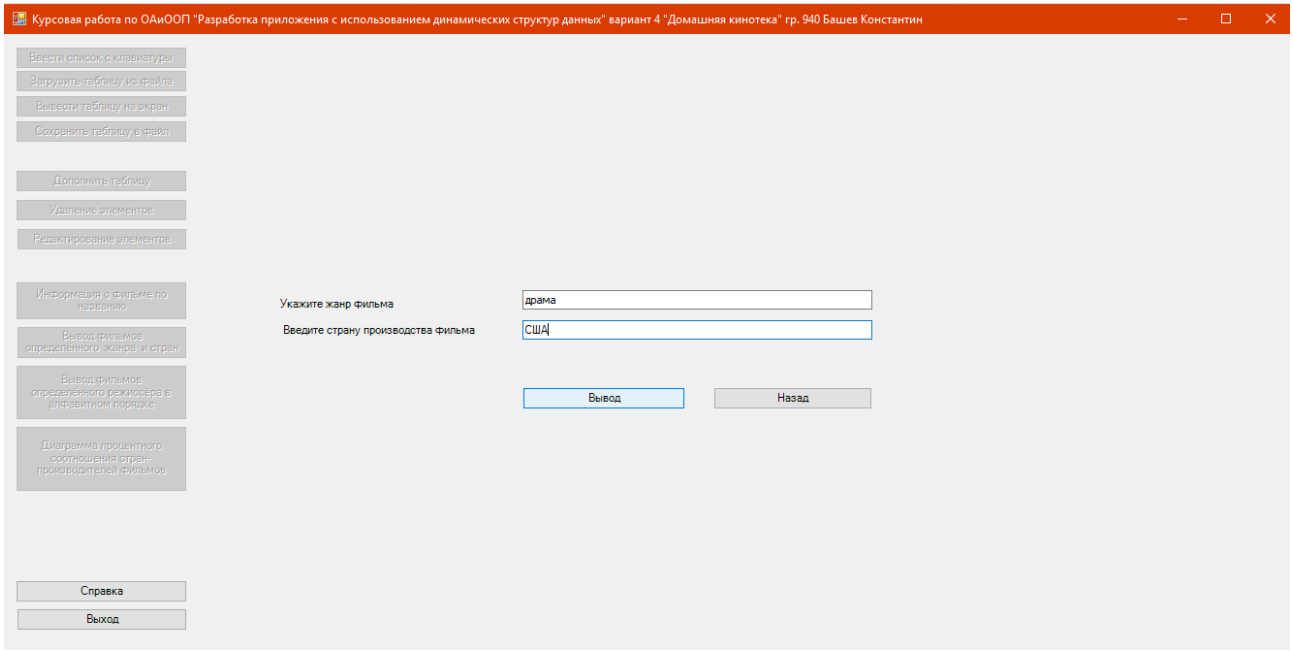

Рисунок 23 — Поиск по стране и жанру

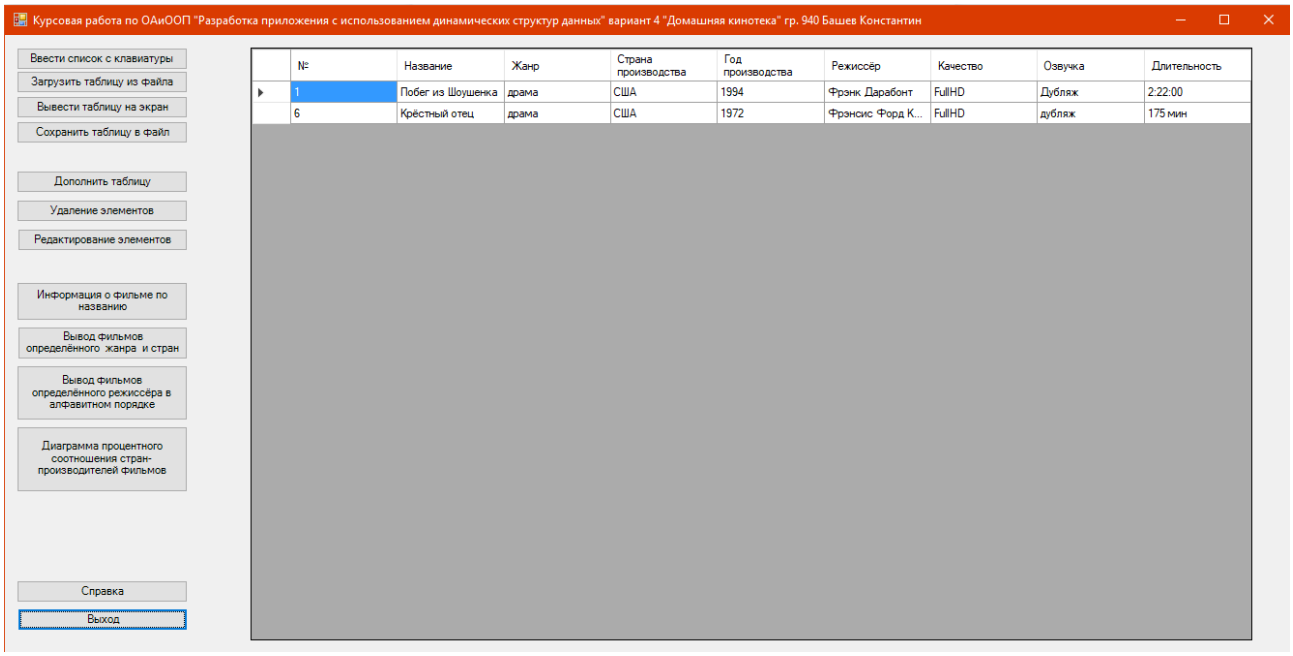

Рисунок 24 — Результаты поиска, все драмы из США

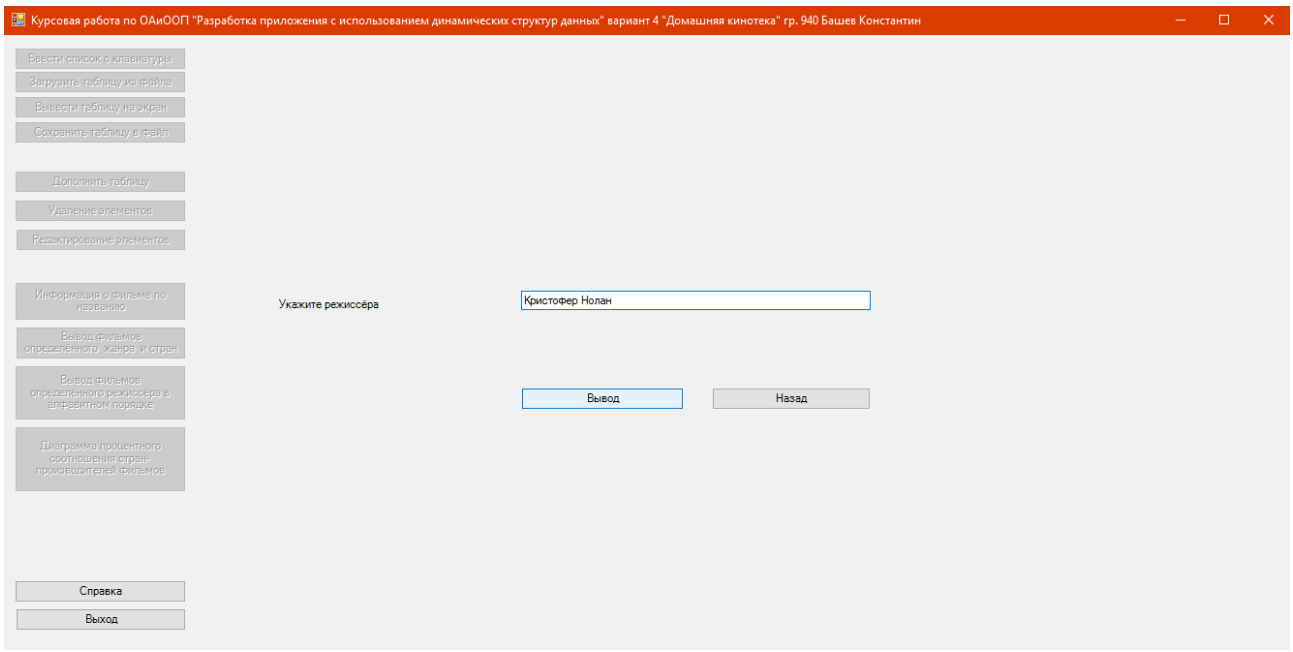

Рисунок 25 — Поиск фильмов определённого режиссёра

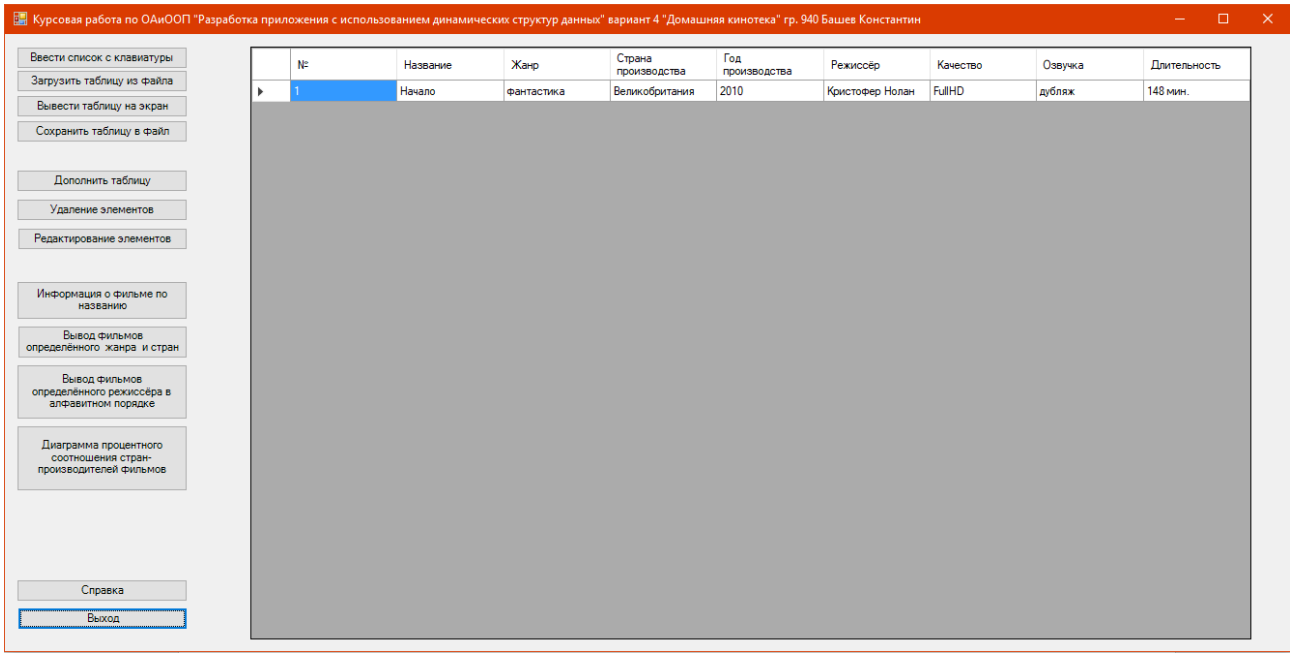

Рисунок 26— Фильмы Кристофера Нолана

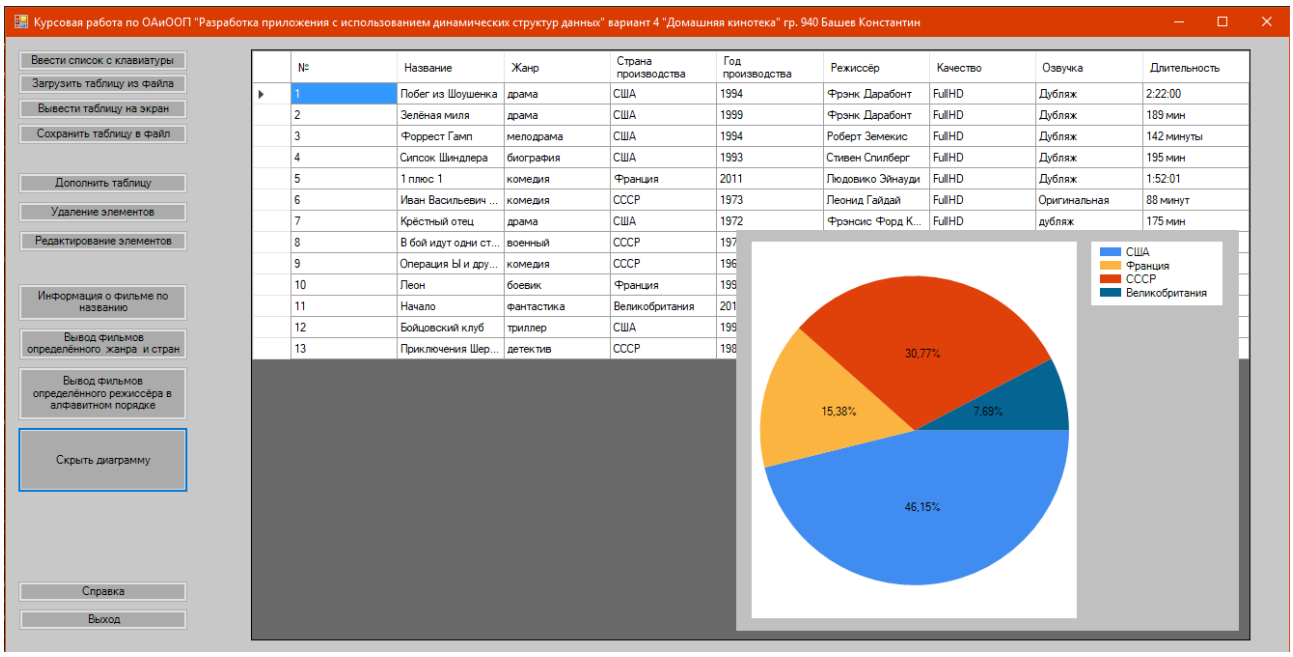

Рисунок 27 — Вывод диаграммы

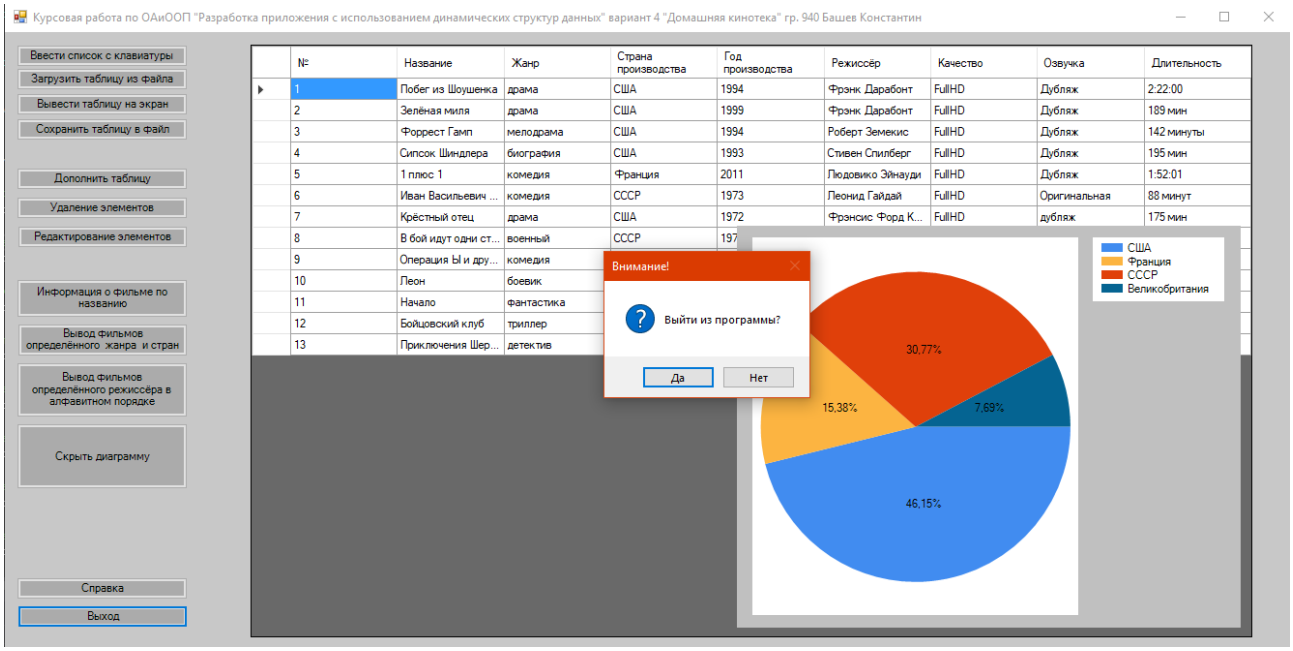

Рисунок 28 — Выход из программы

5. Руководство оператора

5.1. Назначение программы

Программа была разработана в рамках курсового проекта по теме “Разработка приложения с использованием динамических структур данных”, следовательно, в ней должна быть реализована работа со структурой данных “Односвязный список”

В программе реализованы такие функции, как: добавление нового элемента, удаление элемента, редактирование элемента, открытие файла и сохранение таблицы в файл. Кроме того, присутствует возможность поиска и построения диаграммы распределения элементов по жанрам.

5.2. Условия работы программы

Для работы программы требуется следующее аппаратное и программное обеспечение:

- IBM совместимый персональный компьютер
- Операционная система Windows XP, Vista, 7, 8, 10.

5.3. Выполнение программы

При запуске программы пользователю будет показана кнопка с 4 доступными кнопками – ввод и загрузка данных, справка и выход.

После того, как пользователь тем или иным способом заполнит таблицу будет доступен основной функционал приложения – редактирование таблицы, путём изменения, удаления или добавления элементов (Можно выбрать, с каким именно элементом взаимодействовать в специальном окне, которое появится, если включить соответствующий режим. Иначе взаимодействие происходит с последней записью. При редактировании элемента доступен только этот режим – если необходимо изменить последний элемент – необходимо выбрать его).

Также пользователю доступны запросы, описанные в индивидуальном задании, а именно – поиск фильма по названию, выборка фильмов определённой страны и жанра, список фильмов определённого режиссёра. Так же есть возможность сформировать круговую диаграмму соотношения стран-производителей фильмов в кинотеке.

Есть опция «справка».

Все вводимые пользователем данные проверяются – если они не удовлетворяют внутренним критериям – приложение сообщит пользователю об этом.

Завершить выполнение программы можно нажав кнопку «выход» или просто закрыв форму.

5.4. Сообщения оператору

При выполнении программа может выдавать следующие сообщения об ошибках и предупреждения:

При вводе данных в таблицу из файла или с клавиатуры выдаётся предупреждение, что старые данные будут утеряны (см. Рисунок 29, 30)

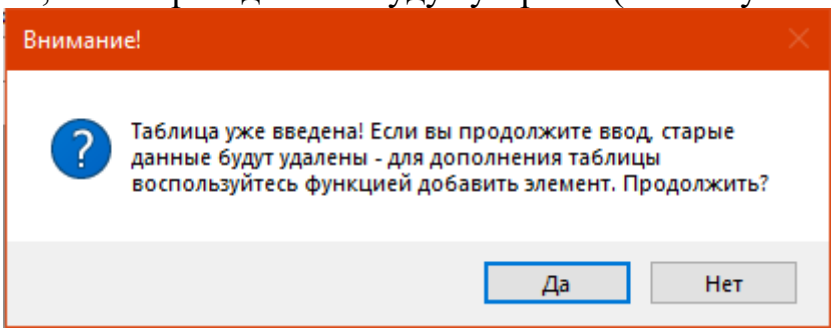

Рисунок 29 — Предупреждение о потере данных 1

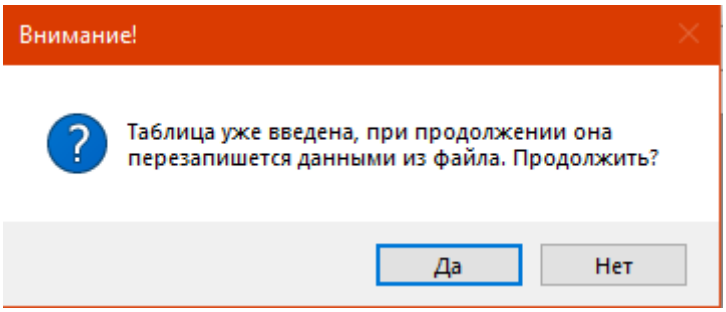

Рисунок 30 — Предупреждение о потере данных 2

Если при вводе каких-либо значений оставить пустые поля или заполнить их пробелами, будут выданы следующие предупреждения (см. Рисунок 31, 32)

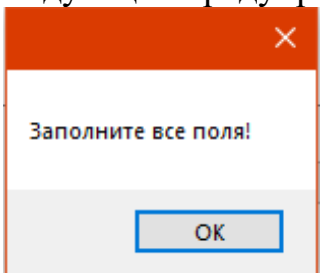

Рисунок 31 — Предупреждение о не заполненных полях 1

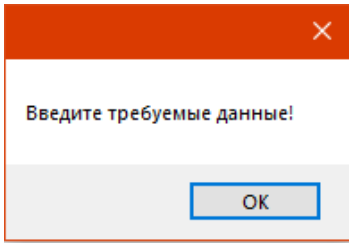

Рисунок 32 — Предупреждение о не заполненных полях 2

Если пользователь поставил на панели галочку «выбрать элемент для…», на сам элемент так и не выбрал, будет показано следующее сообщение (см. Рисунок 33)

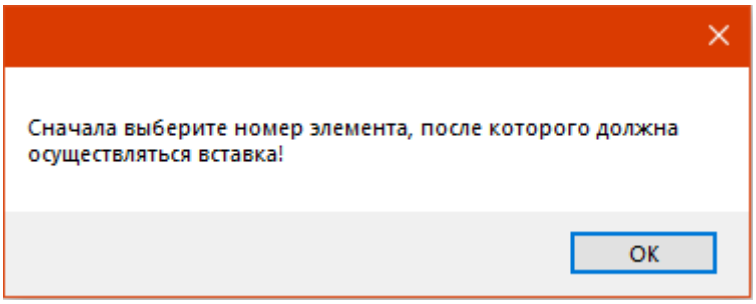

Рисунок 33 — Предупреждение - элемент не выбран

Если пользователь активно пытается ввести некорректные данные в текстовое поле (например, если он несколько раз попытается ввести какую-либо букву в поле для даты), то программа выдаст следующее предупреждение (см. Рисунок 34)

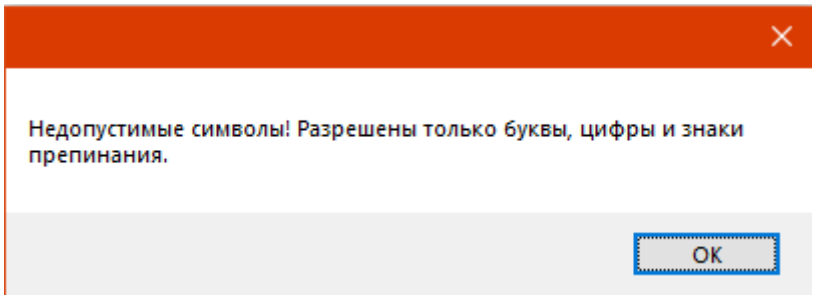

Рисунок 34 — Предупреждение о недопустимых символах

Примечание – содержание описанного выше (см. Рисунок 34) предупреждения варьируется в зависимости от разрешённых к вводу символов для конкретного поля.

Заключение

Как цель курсового проекта была поставлена задача разработки приложения, представляющего собой кинотеку. Ключевой особенностью является то, что необходимо было использовать динамическую структуру данных односвязный список.

Разработанная программа выполняет такие функции как: добавление нового элемента, удаление элемента, редактирование элемента, открытие файла и сохранение таблицы в файл. Кроме того, присутствует возможность поиска и построения диаграммы распределения элементов по странам-производителям.

В разработке мы использовали интегрированную среду разработки Microsoft Visual Studio и платформу .Net. С их помощью мы разработали интерфейс программы.

Поставленную задачу считаю полностью выполненной.

Список использованных источников

1. Интересные вопросы — Хабр Q&A, [электронный ресурс]; URL: https://qna.habr.com/
2. Форум программистов и сисадминов Киберфорум, [электронный ресурс]; URL: https://www.cyberforum.ru/
3. Ответы Mail.ru: Человеческий поиск ответов на любые вопросы, [электронный ресурс]; URL: https://otvet.mail.ru/
4. METANIT.COM – Сайт о программировании, [электронный ресурс]; URL: https://metanit.com/
5. Справочник по C/C++, [электронный ресурс]; URL: http://mycpp.ru/cpp/scpp/
6. Документация по семейству продуктов Visual Studio | Microsoft Docs, [электронный ресурс]; URL: https://docs.microsoft.com/ru-ru/visualstudio/?view=vs-2019
7. Тег C++, [электронный ресурс]; URL: http://blog.kislenko.net/tag.php?id=87 // Блог ПерСа [электронный ресурс]; URL: http://blog.kislenko.net/
8. Курс: Основы алгоритмизации и объектно-ориентированное программирование, [электронный ресурс]; URL: http://cdo.rsreu.ru/course/view.php?id=2266 // Дистанционное обучение РГРТУ, [электронный ресурс]; URL: http://cdo.rsreu.ru/

Приложения

Приложение А — Код файла MovieLibrary.h

```
#pragma once
#include "stdafx.h"
#include <string>
#include <fstream>

using namespace std;

 // Структура для описания одного фильма из кинотеки
struct Movie
{
	string name; // Название
	string genre; // Жанр
	string country; // Страна
	string productionYear; // Год производства
	string producer; // Режиссёр
	string format; // Качество
	string sound; // Звук
	string time; // Продолжительность

	Movie* link; // Указатель на следующий элемент списка
};

 // Класс для работы с кинотекой
 // Реализует основной функционал базы данных в этой программе
class MovieLibrary
{
public:
	MovieLibrary();
	~MovieLibrary();

	int setElAt(Movie, int); // Присваивает указанному элементу переданное значение
	Movie* getElAt(int); // Возвращает указатель на указанный элемент списка
	Movie* getList(); // Возвращает указатель на начало списка
	bool isListEmpty(); // Говорит, пуст ли список

	int saveList(); // Сохранить список в файл
	int loadList(); // Загрузить список из файла
	void setFileName(string); // Установить имя файла

	void createFirstFilm(Movie); // Создать первый элемент списка
	void addFilm(Movie); // Добавить фильм (в конец списка)
	int addFilm(Movie, int); // Добавить фильм после указанного
	int addFirstFilm(Movie); // Добавить фильм в начало списка

	int deleteFilm(int); // Удаляет указанный фильм
	int deleteLastFilm(); // Удаляет последний фильм
	void deleteTable(); // Удаляет таблицу целиком

	void sort(); // Отсортировать список в алфавитном порядке по названию фильма

private:
	string fileName; // Имя файла
	Movie* head; // Указатель на начало списка
	bool isEmpty; // Флаг - пуст ли список

	void swap(Movie*, Movie*); // Поменять два элемента местами
};
```

Приложение Б — Код файла MovieLibrary.cpp

```
#include "stdafx.h"
#include "MovieLibrary.h"
#include "Windows.h"

// Возвращает указатель на указанный элемент списка
Movie * MovieLibrary::getElAt(int index)
{
	if (!isEmpty) // Если список не пуст
	{
		if (index == 1)
			return head; // Возвращаем начало списка (1-ый его эл-т)
		else
		{
			// Ищем указанный элемент
			Movie* current = head;
			int i = 1;
			while (current && i <= index)
			{
				if (i == index) // Нашли
					return current; // Возвращаем его и выходим
				current = current->link;
				i++;
			}
			return NULL; // Возвращаем NULL, если эл-т не найден
		}
	}
	return NULL; // Так же возвращаем NULL, если список пуст
}

// Сохранить список в файл
int MovieLibrary::saveList()
{
	if (isEmpty) // Если список пуст
		return 2; // Возвращаем код ошибки
	else
	{
		ofstream file(fileName); // Открываем файл для записи

		if (!file.is_open()) // Если не удалось открыть файл
			return 1; // Возвращаем код ошибки
		else
		{
			// Записываем список в файл
			Movie* current = head;
			while (current)
			{
				file << current->name << endl;
				file << current->genre << endl;
				file << current->country << endl;
				file << current->productionYear << endl;
				file << current->producer << endl;
				file << current->format << endl;
				file << current->sound << endl;
				if (current->link == NULL)
					file << current->time; // После последнего элемента не
делаем перенос на следующую строку
				else
```

```
                                file << current->time << endl; // После остальных - делаем

                        current = current->link;
                }
                file.close(); // Закрываем файл
        }
    }
    return 0; // Возвращаем 0 при успешной перезаписи файла
}

// Загрузить список из файла
int MovieLibrary::loadList()
{
    ifstream file(fileName); // Открываем файл для чтения

    if (!file.is_open()) // Не удалось открыть файл
    {
        // Закрываем файл и сообщаем об ошибке
        file.close();
        return 1;
    }
    else if (file.peek() == -1 || file.peek() == 0) // Файл пуст
    {
        // Закрываем файл, сообщаем, что он пуст
        file.close();
        return 2;
    }
    else // Данные можно считывать
    {
        // Если список уже заполнен
        if (!(isEmpty && head == NULL))
            this->deleteTable(); // Удаляем его во избежание утечек

        // Считываем файл построчко
        Movie* tmp = NULL;
        Movie* current = head;
        while (!file.eof())
        {
            tmp = new Movie;

            getline(file, tmp->name);
            getline(file, tmp->genre);
            getline(file, tmp->country);
            getline(file, tmp->productionYear);
            getline(file, tmp->producer);
            getline(file, tmp->format);
            getline(file, tmp->sound);
            getline(file, tmp->time);

            tmp->link = NULL;

            if (current != NULL) // При считывании обычных элементов
            {
                current->link = tmp; // Устанавливаем link текущего узла на
только что считанный
                current = tmp; // Делаем только что считанный узел текущим
            }
            else // При считывании первого элемента
                current = head = tmp; // Присваиваем указателю на началов списка
и указателю на текущий элемент
```

```
                                                                //адрес только что считанного
узла
                }

                isEmpty = false; // Устанавливаем флаг (список больше не пуст)

                file.close(); // Закрываем файл
        }

        return 0; // Возвращаем код успешного завершения работы функции
}

// Добавить фильм после указанного
int MovieLibrary::addFilm(Movie m, int indexToAddAfter)
{
        if (isEmpty) // Если список пуст
                return 1; // Возвращаем код ошибки
        else
        {
                int i = 1; // Устанавливаем счётчик на 1
                bool found = false; // Устанавливаем флаг в false
                Movie* current = head; // Устанавливаем указатель на начало списка
                while (current && !found) // Пока не конец списка или пока не найден элемент
                {
                        if (i == indexToAddAfter) // Номер элемента совпал с искомым
                        {
                                // Выделяем память под новый узел списка
                                Movie* tmp = new Movie;
                                tmp->link = current->link; // Устанавливаем link новосозданного
узла на эл-т, следующий за текущим
                                current->link = tmp; // Устанавливаем link текущего узла на
новосозданный эл-т

                                // Инициализируем новосозданный эл-т переданным значением
                                tmp->name = m.name;
                                tmp->genre = m.genre;
                                tmp->country = m.country;
                                tmp->productionYear = m.productionYear;
                                tmp->producer = m.producer;
                                tmp->format = m.format;
                                tmp->sound = m.sound;
                                tmp->time = m.time;

                                // Устанавливаем флаг в true для выхода из цикла
                                found = true;
                        }
                        i++; // Увеличиваем счётчик
                        current = current->link; // Переходим к следующему элементу
                }
                if (!found) // Если элемент так и не был найден
                        return 2; // Возвращаем код ошибки
        }

        return 0; // Сообщаем об успешном завершении функции
}

// Удаляет указанный фильм
int MovieLibrary::deleteFilm(int indexToDelete)
{
        if (isEmpty)
        {
```

```
		return 1; // Возвращаем 1, если список пуст
	}
	else if (head->link == NULL && indexToDelete == 1) // Если в таблице остался один
элемент
		this->deleteTable(); // Удаляем список целиком
	else
	{
		int index = 1; // Устанавливаем счётчик
		if (indexToDelete == 1) // Если нужно удалить первый эл-т и он не единственный
в списке
		{
			Movie* toDelete = head; // Запоминаем адрес начала списка
			head = head->link; // Перемещаем указатель на начало списка на
следующий элемент
			delete toDelete; // Удаляем прошлый элемент
			toDelete = NULL;
			return 0; // Функция успешно отработала
		}
		// Если же удалить нужно не первый эл-т
		index++;
		Movie* current = head; // Устанавливаем указатель на начало списка
		Movie* toDelete = NULL;
		Movie* previous = NULL;
		while (current->link) // Пока не дойдём до предпоследнего элемента
		{
			toDelete = current->link; // Сохраняем адрес следующего за текущим эл-
та
			if (index == indexToDelete) // Если найден эл-т, который требуется
удалить
			{
				current->link = toDelete->link; // Устанавливаем link текущего
эл-та на следующий за тем, который хотим удалить
				delete toDelete; // Удаляем следующий за текущим эл-т
				toDelete = NULL;
				return 0; // Функция успешно отработала
			}
			else // Если эл-т для удаления ещё не найден
			{
				index++; // Увеличиваем счётчик
				previous = current; // Сохраняем адрес текущего элемента
				current = current->link; // Переходи к следующему элементу списка
			}
		}
		// Дошли до предпоследнего эл-та
		if (index == indexToDelete) // нашли элемент для удаления - он последний
		{
			previous->link = NULL; // Устанавливаем link идущего перед текущим узла
на NULL
			delete current; // Удаляем текущий элемент
			current = NULL;
			return 0; // Функция успешно отработала
		}
	}
}

// Отсортировать список в алфавитном порядке по названию фильма
void MovieLibrary::sort()
{
	Movie* current = head;
	if (head == NULL || isEmpty)
		return; // Если список пуст - завершаем работу функции
```

```
	else
	{
		bool isSorted = false; // Флаг (отсортировано) = false
		while (!isSorted) // Пока не отсортировано
		{
			isSorted = true; // Считаем, что список отсортирован
			current = head; // Устанавливаем указатель на начало списка
			while (current->link) // Пока не конец списка
			{
				if (current->name > current->link->name) // Если элементы нужно
поменять местами
				{
					// Меняем их местами
					swap(current, current->link);
					isSorted = false; // Устанавливаем флаг (отсортировано) в
false
				}
				current = current->link; // Переходим к следующему элементу
			}
		}
	}
}
```

Приложение В — Код файла dataCounter.h

```
#pragma once

#include <vector>
#include <string>

#include "MovieLibrary.h" // Подключаем класс для доступа к кинотеке

using namespace std;

// Класс для анализа количества фильмов от каждой страны-производителя
class dataCounter
{
public:
	dataCounter(Movie* ); // Конструктор, принимающий указатель на список фильмов, по
которому должна составляться выборка
	~dataCounter();

	int analyse(); // Обработка данных, составление выборки
	void clear(); // Удаление выборки
	int getAmountOfStats(); // Получить количество стран-производителей в выборке
	string getStatAt(int); // Получить название определённой страны-производителя из
выборки
	int getPointAt(int); // Получить количество фильмов, произведённых в указанной стране

private:
	Movie* head; // Указатель на список фильмов
	int amountOfStats; // Количество стран=производителей в выборке
	vector<string> stats; // Динамический массив, содержащий названия стран-
производителей
	vector<int> points; // Динамический массив, содержащий сведения о количестве фильмов
от каждой страны в выборке

	int thereIsSameStat(string); // Служебная функция для проверки уникальности указанной
страны
                                                    // Возвращает номер этой страны в
выборке, если она уже встречалась
};
```

Приложение Г — Код файла dataCounter.cpp

```
#include "stdafx.h"
#include "dataCounter.h"

// Обработка данных, составление выборки
int dataCounter::analyse()
{
	if(head == NULL)
		return 1; // Выходим, если кинотека пуста
	else
	{
		Movie* current = head; // Устанавливаем указатель на начало кинотеки
		while (current) // Пока не конец кинотеки
		{
			int add = -1;
			if (amountOfStats == 0) // Добавляем первую страну в выборку
			{
				stats.push_back(current->country);
				points.push_back(1);
				amountOfStats++;
			}
			else // Выполняем проверку для последующих эл-тов
			{
				add = thereIsSameStat(current->country); // Встречалась ли эта
страна
				if (add != -1) // Да
					points.at(add)++; // Увеличиваем счётчик фильмов этой
страны
				else // Нет
				{
					stats.push_back(current->country); // Добавляем страну в
выборку
					points.push_back(1);
					amountOfStats++;
				}
			}

			current = current->link; // Устанавливаем указатель на следующий
элемент
		}
	}

	return 0;
}

// Служебная функция для проверки уникальности указанной страны
int dataCounter::thereIsSameStat(string st)
{
	for (int i = 0; i < stats.size(); i++)
		if (stats.at(i) == st)
			return i;
	return -1;
}
```

Приложение Д — Код файла MyForm.h

```
#pragma once

#include "stdafx.h"

#include <iostream>
#include <string>
#include "Windows.h"

#include <msclr\marshal_cppstd.h> // Библиотека для преобразования типов из управляемых в обычные и наоборот (System::String^ в std::string например)

#include "MovieLibrary.h" // Подключаем класс для доступа к кинотеке
#include "dataCounter.h" // Подключаем класс для анализа фильмов каждой страны производителя

namespace CourseWork {

	using namespace std;

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;

	// Создаём перечисление, обозначающее выбранное пользователем действие
	/*
	nul - не выбрано
	inp - ввод данных с клавиатуры
	add - добавить элементы
	del - удалить элементы
	edit - редактировать элементы
	sr1-4 - индивидуальные задания 1-4 соответственно
	*/
	public enum class Action { nul, inp, add, del, edit, sr1, sr2, sr3, sr4 };

	/// <summary>
	/// Сводка для MyForm
	/// </summary>
	// Класс для работы с gui
	public ref class MyForm : public System::Windows::Forms::Form
	{
		// Мои поля
	private:
		int selectedIndex; // Номер выбранного пользователем элемента (для редактирования, дополнения или удаления)
		Action act; // Переменная, хранящяя вседения о выбранном пользователем действии
		MovieLibrary* ML; // Экземпляр класса MovieLibrary для работы со списком фильмов
		Movie* current; // Указатель на текущий элемент списка
		Movie* head; // Указатель на начало списка
		msclr::interop::marshal_context context; // Экземпляр класса marshal_context класса для преобразования типов

		//Мои методы
	private:
		void showList(); // Вывод списка на экран
		void updateDiag(); // Обновить данные для диаграммы
```

```
        // Обработчики событий формы (их прототипы)
            // Обработчики нажатий клавиш в TextBox'ах (текстовых полях) - предназначены
для фильтрации вводимой пользователем информации
private: System::Void fName_KeyPress(System::Object^  sender,
System::Windows::Forms::KeyPressEventArgs^  e);
private: System::Void fGenre_KeyPress(System::Object^  sender,
System::Windows::Forms::KeyPressEventArgs^  e);
private: System::Void fCountry_KeyPress(System::Object^  sender,
System::Windows::Forms::KeyPressEventArgs^  e);
private: System::Void fProdYear_KeyPress(System::Object^  sender,
System::Windows::Forms::KeyPressEventArgs^  e);
private: System::Void fProducer_KeyPress(System::Object^  sender,
System::Windows::Forms::KeyPressEventArgs^  e);
private: System::Void fFormat_KeyPress(System::Object^  sender,
System::Windows::Forms::KeyPressEventArgs^  e);
private: System::Void fSound_KeyPress(System::Object^  sender,
System::Windows::Forms::KeyPressEventArgs^  e);
private: System::Void fTime_KeyPress(System::Object^  sender,
System::Windows::Forms::KeyPressEventArgs^  e);
private: System::Void textBox1_KeyPress(System::Object^  sender,
System::Windows::Forms::KeyPressEventArgs^  e);
private: System::Void textBox2_KeyPress(System::Object^  sender,
System::Windows::Forms::KeyPressEventArgs^  e);

            // Обработчик изменения состояния chexkBox'а - выбирает ли пользователь
конкретный элемент для удаления/добавления/редактирования
                // Срабатывает, если пользователь ставит галочку (заполняет listBox
списком элементов, доступных для удаления/добавления/редактирования)
private: System::Void chooseElNumber_CheckedChanged(System::Object^  sender,
System::EventArgs^  e);

            // Обработчик изменения выделения в listBox (отображает список элементов,
доступных для удаления/добавления/редактирования)
                // Срабатывает, если изменяется выделение - запоминает номер
выделенного элемента
private: System::Void numberOfEl_SelectedIndexChanged(System::Object^  sender,
System::EventArgs^  e);

            // Обработчики нажатий кнопок
                // Подтвердить ввод элемента для
ввода/удаления/добавления/редактирования на специальной панели
private: System::Void inputEl_Click(System::Object^  sender, System::EventArgs^  e);

                // Нажатие кнопок вывода на экран, сохранения и загрузки а так же ввода
списка с клавиатуры соответственно
private: System::Void outputList_Click(System::Object^  sender, System::EventArgs^  e);
private: System::Void rewriteFile_Click(System::Object^  sender, System::EventArgs^  e);
private: System::Void readFile_Click(System::Object^  sender, System::EventArgs^  e);
private: System::Void inputList_Click(System::Object^  sender, System::EventArgs^  e);

                // Кнопка прекратить действие на специальной панели (для добавления,
удаления, ввода с клавиатуры и редактирования)
private: System::Void stopInputList_Click(System::Object^  sender, System::EventArgs^  e);

                // Нажатие кнопок дополнить, удалить и редактировать элемент таблицы
соответственно
private: System::Void addToList_Click(System::Object^  sender, System::EventArgs^  e);
private: System::Void deleteElement_Click(System::Object^  sender, System::EventArgs^  e);
private: System::Void editElement_Click(System::Object^  sender, System::EventArgs^  e);

                // Нажатие кнопок для выполнения индивидуальных запросов
```

```
private: System::Void specialRequest1_Click(System::Object^  sender, System::EventArgs^  e);
private: System::Void specialRequest2_Click(System::Object^  sender, System::EventArgs^  e);
private: System::Void specialRequest3_Click(System::Object^  sender, System::EventArgs^  e);
private: System::Void specialRequest4_Click(System::Object^  sender, System::EventArgs^  e);

                    // Нажатие кнопок подтверждения и отмены на специальной панели при
выполнении индивидуальных запросов
private: System::Void confirmButton_Click(System::Object^  sender, System::EventArgs^  e);
private: System::Void cancelButton_Click(System::Object^  sender, System::EventArgs^  e);

                    // Нажатие кнопки выхода
private: System::Void exitButton_Click(System::Object^  sender, System::EventArgs^  e);

                    // Вывод подсказки
private: System::Void helpButton_Click(System::Object^  sender, System::EventArgs^  e);
};
}
```

Приложение Е — Код файла MyForm.cpp

```
#include "stdafx.h"
#include "MyForm.h"

// Нажата кнопка "ввод" на специальной панели
System::Void CourseWork::MyForm::inputEl_Click(System::Object ^ sender, System::EventArgs ^
e)
{
	head = ML->getList(); // Получаем данные

	if (act == Action::inp) // Если панель вызвана с целью ввода списка с клавиатуры
	{
		Movie m; // Временная переменная

		// Убираем лишние пробелы из текстовых окон
		String^ text = fName->Text;
		while (text->Contains("  "))
			text = text->Replace("  ", " ");
		fName->Text = text;

		text = fGenre->Text;
		while (text->Contains("  "))
			text = text->Replace("  ", " ");
		fGenre->Text = text;

		text = fCountry->Text;
		while (text->Contains("  "))
			text = text->Replace("  ", " ");
		fCountry->Text = text;

		text = fProducer->Text;
		while (text->Contains("  "))
			text = text->Replace("  ", " ");
		fProducer->Text = text;

		text = fFormat->Text;
		while (text->Contains("  "))
			text = text->Replace("  ", " ");
		fFormat->Text = text;

		text = fSound->Text;
		while (text->Contains("  "))
			text = text->Replace("  ", " ");
		fSound->Text = text;

		text = fTime->Text;
		while (text->Contains("  "))
			text = text->Replace("  ", " ");
		fTime->Text = text;

		// Проверяем введённые пользователем данные
		if (
			fName->Text == "Введите название" || fName->Text == "" || fName->Text
== " " ||
			fGenre->Text == "Введите жанр" || fGenre->Text == "" || fGenre->Text ==
" " ||
			fCountry->Text == "Введите страну производства" || fCountry->Text == ""
|| fCountry->Text == " " ||
			fProdYear->Text == "Введите год производства" || fProdYear->Text == ""
|| fProdYear->Text == " " ||
```

```
                fProducer->Text == "Введите режиссёра" || fProducer->Text == "" ||
fProducer->Text == " " ||
                fFormat->Text == "Введите качество" || fFormat->Text == "" || fFormat-
>Text == " " ||
                fSound->Text == "Введите озвучку" || fSound->Text == "" || fSound->Text
== " " ||
                fTime->Text == "Введите длительность" || fTime->Text == "" || fTime-
>Text == " ")
            {
                MessageBox::Show("Заполните все поля!");
                return System::Void();
            }

            // Преобразуем типы из управляемого Text в стандартный std::string
            m.name = context.marshal_as<std::string>(fName->Text);
            m.genre = context.marshal_as<std::string>(fGenre->Text);
            m.country = context.marshal_as<std::string>(fCountry->Text);
            m.productionYear = context.marshal_as<std::string>(fProdYear->Text);
            m.producer = context.marshal_as<std::string>(fProducer->Text);
            m.format = context.marshal_as<std::string>(fFormat->Text);
            m.sound = context.marshal_as<std::string>(fSound->Text);
            m.time = context.marshal_as<std::string>(fTime->Text);

            // В зависимости от того, пуст ли список
            if (ML->isListEmpty())
                ML->createFirstFilm(m); // Создаём первый элемент
            else
                ML->addFilm(m); // Или добавляем элемент в конец

            // Выводим текст-приглашение на текстовые поля
            fName->Text = "Введите название";
            fGenre->Text = "Введите жанр";
            fCountry->Text = "Введите страну производства";
            fProdYear->Text = "Введите год производства";
            fProducer->Text = "Введите режиссёра";
            fFormat->Text = "Введите качество";
            fSound->Text = "Введите озвучку";
            fTime->Text = "Введите длительность";
        }
else if (act == Action::add) // Если панель вызвана с целью добавить элемент в таблицу
        {
            Movie m; // Создаём временную переменную

            // Удаляем лишние проблелы
            String^ text = fName->Text;
            while (text->Contains("  "))
                text = text->Replace("  ", " ");
            fName->Text = text;

            text = fGenre->Text;
            while (text->Contains("  "))
                text = text->Replace("  ", " ");
            fGenre->Text = text;

            text = fCountry->Text;
            while (text->Contains("  "))
                text = text->Replace("  ", " ");
            fCountry->Text = text;

            text = fProducer->Text;
            while (text->Contains("  "))
```

```
		text = text->Replace("  ", " ");
	fProducer->Text = text;

	text = fFormat->Text;
	while (text->Contains("  "))
		text = text->Replace("  ", " ");
	fFormat->Text = text;

	text = fSound->Text;
	while (text->Contains("  "))
		text = text->Replace("  ", " ");
	fSound->Text = text;

	text = fTime->Text;
	while (text->Contains("  "))
		text = text->Replace("  ", " ");
	fTime->Text = text;

	// Проверяем введённые данные
	if (
		fName->Text == "Введите название" || fName->Text == "" || fName->Text
== " " ||
		fGenre->Text == "Введите жанр" || fGenre->Text == "" || fGenre->Text ==
" " ||
		fCountry->Text == "Введите страну производства" || fCountry->Text == ""
|| fCountry->Text == " " ||
		fProdYear->Text == "Введите год производства" || fProdYear->Text == ""
|| fProdYear->Text == " " ||
		fProducer->Text == "Введите режиссёра" || fProducer->Text == "" ||
fProducer->Text == " " ||
		fFormat->Text == "Введите качество" || fFormat->Text == "" || fFormat-
>Text == " " ||
		fSound->Text == "Введите озвучку" || fSound->Text == "" || fSound->Text
== " " ||
		fTime->Text == "Введите длительность" || fTime->Text == "" || fTime-
>Text == " ")
	{
		MessageBox::Show("Заполните все поля!");
		return System::Void();
	}

	// Преобразуем их
	m.name = context.marshal_as<std::string>(fName->Text);
	m.genre = context.marshal_as<std::string>(fGenre->Text);
	m.country = context.marshal_as<std::string>(fCountry->Text);
	m.productionYear = context.marshal_as<std::string>(fProdYear->Text);
	m.producer = context.marshal_as<std::string>(fProducer->Text);
	m.format = context.marshal_as<std::string>(fFormat->Text);
	m.sound = context.marshal_as<std::string>(fSound->Text);
	m.time = context.marshal_as<std::string>(fTime->Text);

	// Добавляем элемент в список
	if (!chooseElNumber->Checked) // Если пользователь не выбрал конкретное место
		if (!ML->isListEmpty()) // Добавляем в конец
		{
			ML->addFilm(m);
			MessageBox::Show("Элемент добавлен в конец списка!");
		}
		else
			MessageBox::Show("Список пуст! Прежде чем дополнять таблицу
введите её с клавиатуры или загрузите из файла!");
```

```
		else
			if (!ML->isListEmpty())
			{
				if (selectedIndex > 1) // Если пользователь выбрал, после какого
из элементов должен быть добавлен новый
				{
					if (ML->addFilm(m, selectedIndex - 1) == 0) // Добавляем
						MessageBox::Show("Элемент успешно добавлен");
				}
				else if (selectedIndex == 1) // Если пользователь хочет добавить
элемент в начало списка
				{
					if (ML->addFirstFilm(m) == 0) // Добавляем
					{
						head = ML->getList();
						MessageBox::Show("Элемент успешно добавлен в начало
списка");
					}

				}
				else // Если пользователь хочет добавить элемент в конкретное
место (поставлена галочка в checkBox), но не указал куда
					MessageBox::Show("Сначала выберите номер элемента, после
которого должна осуществляться вставка!");
			}
			else
				MessageBox::Show("Список пуст! Прежде чем дополнять таблицу
введите её с клавиатуры или загрузите из файла!");

		// Заново инициализируем некоторые компоненты после изменения списка
		current = ML->getList();
		int i = 1;
		numberOfEl->ClearSelected();
		numberOfEl->Items->Clear();
		numberOfEl->BeginUpdate();
		numberOfEl->Items->Add("Добавить в начало списка");
		while (current)
		{
			numberOfEl->Items->Add(i.ToString() + ") " +
context.marshal_as<String^>(current->name));
			current = current->link;
			i++;
		}
		numberOfEl->EndUpdate();

		selectedIndex = 0;

		fName->Text = "Введите название";
		fGenre->Text = "Введите жанр";
		fCountry->Text = "Введите страну производства";
		fProdYear->Text = "Введите год производства";
		fProducer->Text = "Введите режиссёра";
		fFormat->Text = "Введите качество";
		fSound->Text = "Введите озвучку";
		fTime->Text = "Введите длительность";
	}
else if (act == Action::del) // Если панель вызвана с целью удалить элемент
	{
		if (!chooseElNumber->Checked) // Пользователь не выбрал конкретный элемент
			if (!ML->isListEmpty())
			{
```

```
                        ML->deleteLastFilm(); // Удаляем последний
                        MessageBox::Show("Последний элемент успешно удалён");
                        head = ML->getList();
                }
                else
                {
                        MessageBox::Show("Список пуст! Прежде чем дополнять таблицу
введите её с клавиатуры или загрузите из файла!");
                        head = NULL;
                }
        else
                if (!ML->isListEmpty())
                {
                        if (selectedIndex > 0) // Пользователь выбрал элемент для
удаления
                        {
                                if (ML->deleteFilm(selectedIndex) == 0) // Удаляем его
                                        MessageBox::Show("Элемент успешно удалён");
                                head = ML->getList();
                        }
                        else
                                MessageBox::Show("Сначала выберите номер элемента, который
нужно удалить!");
                }
                else
                {
                        MessageBox::Show("Список пуст! Прежде чем дополнять таблицу
введите её с клавиатуры или загрузите из файла!");
                        head = NULL;
                }

        // Обновляем данные после удаления элементов
        int i = 1;
        numberOfEl->ClearSelected();
        numberOfEl->Items->Clear();
        numberOfEl->BeginUpdate();
        while (current)
        {
                numberOfEl->Items->Add(i.ToString() + ") " +
context.marshal_as<String^>(current->name));
                current = current->link;
                i++;
        }
        numberOfEl->EndUpdate();

        selectedIndex = 0;
    }
else if (act == Action::edit) // Панель вызвана с целью отредактироваль элемент
    {
        if (selectedIndex == 0) // Просим пользователя выбрать элемент для
редактирования, если он этого не сделал
                MessageBox::Show("Сначала выберите элемент, который хотите
отредактировать");
        else
        {
                Movie m;

                // Убираем пробелы
                String^ text = fName->Text;
                while (text->Contains("  "))
                        text = text->Replace("  ", " ");
```

```
                    fName->Text = text;

                    text = fGenre->Text;
                    while (text->Contains("  "))
                           text = text->Replace("  ", " ");
                    fGenre->Text = text;

                    text = fCountry->Text;
                    while (text->Contains("  "))
                           text = text->Replace("  ", " ");
                    fCountry->Text = text;

                    text = fProducer->Text;
                    while (text->Contains("  "))
                           text = text->Replace("  ", " ");
                    fProducer->Text = text;

                    text = fFormat->Text;
                    while (text->Contains("  "))
                           text = text->Replace("  ", " ");
                    fFormat->Text = text;

                    text = fSound->Text;
                    while (text->Contains("  "))
                           text = text->Replace("  ", " ");
                    fSound->Text = text;

                    text = fTime->Text;
                    while (text->Contains("  "))
                           text = text->Replace("  ", " ");
                    fTime->Text = text;

                    // Проверяем ввод
                    if (
                           fName->Text == "Введите название" || fName->Text == "" || fName-
>Text == " " ||
                           fGenre->Text == "Введите жанр" || fGenre->Text == "" || fGenre-
>Text == " " ||
                           fCountry->Text == "Введите страну производства" || fCountry->Text
== "" || fCountry->Text == " " ||
                           fProdYear->Text == "Введите год производства" || fProdYear->Text
== "" || fProdYear->Text == " " ||
                           fProducer->Text == "Введите режиссёра" || fProducer->Text == ""
|| fProducer->Text == " " ||
                           fFormat->Text == "Введите качество" || fFormat->Text == "" ||
fFormat->Text == " " ||
                           fSound->Text == "Введите озвучку" || fSound->Text == "" ||
fSound->Text == " " ||
                           fTime->Text == "Введите длительность" || fTime->Text == "" ||
fTime->Text == " ")
                    {
                           MessageBox::Show("Заполните все поля!");
                           return System::Void();
                    }

                    // Преобразуем данные
                    m.name = context.marshal_as<std::string>(fName->Text);
                    m.genre = context.marshal_as<std::string>(fGenre->Text);
                    m.country = context.marshal_as<std::string>(fCountry->Text);
                    m.productionYear = context.marshal_as<std::string>(fProdYear->Text);
                    m.producer = context.marshal_as<std::string>(fProducer->Text);
```

```
                        m.format = context.marshal_as<std::string>(fFormat->Text);
                        m.sound = context.marshal_as<std::string>(fSound->Text);
                        m.time = context.marshal_as<std::string>(fTime->Text);

                        // Присваиваем выбранному элементу введённое значение
                        ML->setElAt(m, selectedIndex);
                        MessageBox::Show("Элемент отредактирован");

                        // Обновляем данные
                        selectedIndex = 0;

                        fName->Text = "Введите название";
                        fGenre->Text = "Введите жанр";
                        fCountry->Text = "Введите страну производства";
                        fProdYear->Text = "Введите год производства";
                        fProducer->Text = "Введите режиссёра";
                        fFormat->Text = "Введите качество";
                        fSound->Text = "Введите озвучку";
                        fTime->Text = "Введите длительность";

                        current = ML->getList();
                        int i = 1;
                        numberOfEl->ClearSelected();
                        numberOfEl->Items->Clear();
                        numberOfEl->BeginUpdate();
                        while (current)
                        {
                                numberOfEl->Items->Add(i.ToString() + ") " +
context.marshal_as<String^>(current->name));
                                current = current->link;
                                i++;
                        }
                        numberOfEl->EndUpdate();

                        fName->Enabled = false;
                        fGenre->Enabled = false;
                        fCountry->Enabled = false;
                        fProdYear->Enabled = false;
                        fProducer->Enabled = false;
                        fFormat->Enabled = false;
                        fSound->Enabled = false;
                        fTime->Enabled = false;
                }
        }

        // Обновляем данные ещё раз
        head = ML->getList();
        current = head;
        chooseElNumber->Checked = false;
}

// Нажата кнопка "Вывести таблицу на экран"
System::Void CourseWork::MyForm::outputList_Click(System::Object ^ sender, System::EventArgs
^ e)
{
        current = head;
        if (head == NULL) // Если табица не пуста
                if (ML->isListEmpty())
                {
                        MessageBox::Show("Список пуст! Прежде, чем выводить его на экран -
заполните таблицу вручную или загрузите из файла.");
```

```
                return System::Void();
            }
            else
            {
                // Получаем данные о кинотеке
                head = ML->getList();
                current = head;
            }

        // Выводим её на экран
        this->showList();

        // обновляем данные
        current = head;

        return System::Void();
}

// Пользователь запрашивает ввод данных с клавиатуры
System::Void CourseWork::MyForm::inputList_Click(System::Object ^ sender, System::EventArgs
^ e)
{
        if (!ML->isListEmpty())
        {
            //Предупреждаем, что текущая таблица удалится
            System::Windows::Forms::DialogResult ee;
            ee = MessageBox::Show("Таблица уже введена! Если вы продолжите ввод, старые
данные будут удалены - для дополнения таблицы воспользуйтесь функцией ""добавить элемент"".
Продолжить?","Внимание!", MessageBoxButtons::YesNo, MessageBoxIcon::Question);
            if (ee == System::Windows::Forms::DialogResult::Yes)
            {
                ML->deleteTable();
            }
            else
                return System::Void();
        }

        // Устанавливаем режим ввода с клавиатуры
        act = Action::inp;

        // Выводим на форму нужные для ввода элементы
        inputListPanel->Enabled = true;
        inputListPanel->Visible = true;

        inputEl->Text = "Добавить элемент";

        // Скрываем другие графические элементы
        dataGridViewForList->Visible = false;

        inputList->Enabled = false;
        outputList->Enabled = false;
        readFile->Enabled = false;
        rewriteFile->Enabled = false;

        addToList->Enabled = false;
        deleteElement->Enabled = false;
        editElement->Enabled = false;

        specialRequest1->Enabled = false;
        specialRequest2->Enabled = false;
        specialRequest3->Enabled = false;
```

```
	specialRequest4->Enabled = false;

	return System::Void();
}

// Нажата кнопка "назад" на специальной панели
System::Void CourseWork::MyForm::stopInputList_Click(System::Object ^ sender,
System::EventArgs ^ e)
{
	// Прячем элементы формы
	inputListPanel->Enabled = false;
	inputListPanel->Visible = false;

	chooseElNumber->Visible = false;
	chooseElNumber->Enabled = false;
	labelToPurpose->Visible = false;
	labelToPurpose->Enabled = false;

	chooseElNumber->Checked = false;

	numberOfEl->Visible = false;
	numberOfEl->Enabled = false;

	inputList->Enabled = true;
	readFile->Enabled = true;
	if (!ML->isListEmpty()) // Если Таблица заполнена - показываем кнопки для
взаимодействия с таблицей и саму таблицу
	{
		dataGridViewForList->Visible = true;

		outputList->Enabled = true;
		rewriteFile->Enabled = true;
		addToList->Enabled = true;
		deleteElement->Enabled = true;
		editElement->Enabled = true;

		specialRequest1->Enabled = true;
		specialRequest2->Enabled = true;
		specialRequest3->Enabled = true;
		specialRequest4->Enabled = true;
	}

	// Включаем другие элементы формы (они находятся на специальных панелях и всё равно
невидимы)
	// Однако некоторые функции могли их спрятать - нужно вывести их, чтобы сделать
возможной работу других функций
	fName->Enabled = true;
	fGenre->Enabled = true;
	fCountry->Enabled = true;
	fProdYear->Enabled = true;
	fProducer->Enabled = true;
	fFormat->Enabled = true;
	fSound->Enabled = true;
	fTime->Enabled = true;

	fName->Visible = true;
	fGenre->Visible = true;
	fCountry->Visible = true;
	fProdYear->Visible = true;
	fProducer->Visible = true;
	fFormat->Visible = true;
```

```
        fSound->Visible = true;
        fTime->Visible = true;

        label1->Visible = true;
        label2->Visible = true;
        label3->Visible = true;
        label4->Visible = true;
        label5->Visible = true;
        label6->Visible = true;
        label7->Visible = true;
        label8->Visible = true;

        // Устанавливаем режим ожидания команды
        act = Action::nul;

        // Обновляем данные
        fName->Text = "Введите название";
        fGenre->Text = "Введите жанр";
        fCountry->Text = "Введите страну производства";
        fProdYear->Text = "Введите год производства";
        fProducer->Text = "Введите режиссёра";
        fFormat->Text = "Введите качество";
        fSound->Text = "Введите озвучку";
        fTime->Text = "Введите длительность";



        numberOfEl->ClearSelected();
        numberOfEl->Items->Clear();

        selectedIndex = 0;

        // Выводим на экран таблицу и обновляем данные для диаграммы
        this->showList();
        this->updateDiag();

        return System::Void();
}

// Нажате кнопка подтверждения на специальной панели для индивидуальных запросов
System::Void CourseWork::MyForm::confirmButton_Click(System::Object ^ sender,
System::EventArgs ^ e)
{
        // Очищаем текстовые поля от лишних пробелов
        String^ text = textBox1->Text;
        while (text->Contains("  "))
                text = text->Replace("  ", " ");
        textBox1->Text = text;

        text = textBox2->Text;
        while (text->Contains("  "))
                text = text->Replace("  ", " ");
        textBox2->Text = text;

        // Следим, чтобы таблица не была пуста
        if (ML->isListEmpty() == true)
        {
                MessageBox::Show("Таблица пуста! Сначала заполните её!");
                return System::Void();
        }
```

```
	// Следим, чтобы пользователь ввёл все необходимые данные
	if ( ( String::IsNullOrEmpty(textBox1->Text->ToString()) || textBox1->Text == " " )
||
		act == Action::sr2 &&  ( String::IsNullOrEmpty(textBox2->Text->ToString()) ||
textBox2->Text == " ") )
	{
		MessageBox::Show("Введите требуемые данные!");
		return System::Void();
	}
	// В зависимости от запроса выполняем различные действия
if (act == Action::sr1) // Вывод на экран информации о фильме по названию
	{
		// Работа функции аналогична работе метода showList()

		head = ML->getList();
		current = head;

		if (ML->isListEmpty())
			return;

		dataGridViewForList->DataSource = NULL;

		int i = 1;

		DataTable ^table; //Невидимая таблица данных
		DataColumn ^column; //Столбец таблицы
		DataRow ^row; //Строка таблицы

		//Создаем таблицу данных
		table = gcnew DataTable();

		//Названия столбцов
		String^ Column1 = "№";
		String^ Column2 = "Название";
		String^ Column3 = "Жанр";
		String^ Column4 = "Страна производства";
		String^ Column5 = "Год производства";
		String^ Column6 = "Режиссёр";
		String^ Column7 = "Качество";
		String^ Column8 = "Озвучка";
		String^ Column9 = "Длительность";

		//Создание и добавление столбцов
		column = gcnew DataColumn(Column1, Int32::typeid);
		table->Columns->Add(column);
		column = gcnew DataColumn(Column2, String::typeid);
		table->Columns->Add(column);
		column = gcnew DataColumn(Column3, String::typeid);
		table->Columns->Add(column);
		column = gcnew DataColumn(Column4, String::typeid);
		table->Columns->Add(column);
		column = gcnew DataColumn(Column5, String::typeid);
		table->Columns->Add(column);
		column = gcnew DataColumn(Column6, String::typeid);
		table->Columns->Add(column);
		column = gcnew DataColumn(Column7, String::typeid);
		table->Columns->Add(column);
		column = gcnew DataColumn(Column8, String::typeid);
		table->Columns->Add(column);
		column = gcnew DataColumn(Column9, String::typeid);
		table->Columns->Add(column);
```

```
		while (current != NULL) {
			//Заполняем строчку таблицы
			row = table->NewRow();
			row[Column1] = i;
			row[Column2] = context.marshal_as<String^>(current->name);
			row[Column3] = context.marshal_as<String^>(current->genre);
			row[Column4] = context.marshal_as<String^>(current->country);
			row[Column5] = context.marshal_as<String^>(current->productionYear);
			row[Column6] = context.marshal_as<String^>(current->producer);
			row[Column7] = context.marshal_as<String^>(current->format);
			row[Column8] = context.marshal_as<String^>(current->sound);
			row[Column9] = context.marshal_as<String^>(current->time);

			// Единственное отличие - в таблицу добавляются лишь те элементы, где
название фильма совпадает в введённым пользователем
			if (textBox1->Text->ToString() == row[Column2]->ToString())
				table->Rows->Add(row);
			i++;
			current = current->link;
		}

		//Отображаем таблицу в визуальном компоненте
		dataGridViewForList->DataSource = table;

		current = head;
	}
	if (act == Action::sr2) // Вывод на экран фильмов определённого жанра и страны
	{
		// Работа данной функции аналогична работе предыдущей

		head = ML->getList();
		current = head;

		if (ML->isListEmpty())
			return;

		dataGridViewForList->DataSource = NULL;

		int i = 1;

		DataTable ^table; //Невидимая таблица данных
		DataColumn ^column; //Столбец таблицы
		DataRow ^row; //Строка таблицы

		//Создаем таблицу данных
		table = gcnew DataTable();

		//Названия столбцов
		String^ Column1 = "№";
		String^ Column2 = "Название";
		String^ Column3 = "Жанр";
		String^ Column4 = "Страна производства";
		String^ Column5 = "Год производства";
		String^ Column6 = "Режиссёр";
		String^ Column7 = "Качество";
		String^ Column8 = "Озвучка";
		String^ Column9 = "Длительность";

		//Создание и добавление столбцов
		column = gcnew DataColumn(Column1, Int32::typeid);
```

```
			table->Columns->Add(column);
			column = gcnew DataColumn(Column2, String::typeid);
			table->Columns->Add(column);
			column = gcnew DataColumn(Column3, String::typeid);
			table->Columns->Add(column);
			column = gcnew DataColumn(Column4, String::typeid);
			table->Columns->Add(column);
			column = gcnew DataColumn(Column5, String::typeid);
			table->Columns->Add(column);
			column = gcnew DataColumn(Column6, String::typeid);
			table->Columns->Add(column);
			column = gcnew DataColumn(Column7, String::typeid);
			table->Columns->Add(column);
			column = gcnew DataColumn(Column8, String::typeid);
			table->Columns->Add(column);
			column = gcnew DataColumn(Column9, String::typeid);
			table->Columns->Add(column);

			while (current != NULL) {
				//Заполняем строчку таблицы
				row = table->NewRow();
				row[Column1] = i;
				row[Column2] = context.marshal_as<String^>(current->name);
				row[Column3] = context.marshal_as<String^>(current->genre);
				row[Column4] = context.marshal_as<String^>(current->country);
				row[Column5] = context.marshal_as<String^>(current->productionYear);
				row[Column6] = context.marshal_as<String^>(current->producer);
				row[Column7] = context.marshal_as<String^>(current->format);
				row[Column8] = context.marshal_as<String^>(current->sound);
				row[Column9] = context.marshal_as<String^>(current->time);

				// Единственное отличие - более строгое условие - должны совпасть и
страна и жанр
				if (textBox1->Text->ToString() == row[Column3]->ToString() && textBox2-
>Text->ToString() == row[Column4]->ToString())
					table->Rows->Add(row);
				i++;
				current = current->link;
			}

			//Отображаем таблицу в визуальном компоненте
			dataGridViewForList->DataSource = table;

			current = head;
		}
		if (act == Action::sr3) // Вывод на экран фильмов определённого режиссёра в
алфавитном порядке
		{
			// Создаём новую временную кинотеку
			MovieLibrary* tmp = new MovieLibrary;
			head = ML->getList();
			current = head;

			std::string standart = context.marshal_as<std::string>(textBox1->Text);

			// Добавляем туда элементы с требуемым именем режиссёра
			while (current)
			{
				if (current->producer == standart)
				{
					if (tmp->isListEmpty())
```

```
                tmp->createFirstFilm(*current);
            else
                tmp->addFilm(*current);
        }

        current = current->link;
}

// Выводим резальтат на экран
if (tmp->isListEmpty())
        MessageBox::Show("Фильмов этого режиссёра не найдено!");
else
{
        // Сортируем временный список
        tmp->sort();

        // Далее функция работает идентично предыдущим
        dataGridViewForList->DataSource = NULL;

        int i = 1;

        DataTable ^table; //Невидимая таблица данных
        DataColumn ^column; //Столбец таблицы
        DataRow ^row; //Строка таблицы

        //Создаем таблицу данных
        table = gcnew DataTable();

        //Названия столбцов
        String^ Column1 = "№";
        String^ Column2 = "Название";
        String^ Column3 = "Жанр";
        String^ Column4 = "Страна производства";
        String^ Column5 = "Год производства";
        String^ Column6 = "Режиссёр";
        String^ Column7 = "Качество";
        String^ Column8 = "Озвучка";
        String^ Column9 = "Длительность";

        //Создание и добавление столбцов
        column = gcnew DataColumn(Column1, Int32::typeid);
        table->Columns->Add(column);
        column = gcnew DataColumn(Column2, String::typeid);
        table->Columns->Add(column);
        column = gcnew DataColumn(Column3, String::typeid);
        table->Columns->Add(column);
        column = gcnew DataColumn(Column4, String::typeid);
        table->Columns->Add(column);
        column = gcnew DataColumn(Column5, String::typeid);
        table->Columns->Add(column);
        column = gcnew DataColumn(Column6, String::typeid);
        table->Columns->Add(column);
        column = gcnew DataColumn(Column7, String::typeid);
        table->Columns->Add(column);
        column = gcnew DataColumn(Column8, String::typeid);
        table->Columns->Add(column);
        column = gcnew DataColumn(Column9, String::typeid);
        table->Columns->Add(column);

        head = tmp->getList();
        current = head;
```

```
                    while (current != NULL) {
                        //Заполняем строчку таблицы
                        row = table->NewRow();
                        row[Column1] = i;
                        row[Column2] = context.marshal_as<String^>(current->name);
                        row[Column3] = context.marshal_as<String^>(current->genre);
                        row[Column4] = context.marshal_as<String^>(current->country);
                        row[Column5] = context.marshal_as<String^>(current-
>productionYear);
                        row[Column6] = context.marshal_as<String^>(current->producer);
                        row[Column7] = context.marshal_as<String^>(current->format);
                        row[Column8] = context.marshal_as<String^>(current->sound);
                        row[Column9] = context.marshal_as<String^>(current->time);

                        table->Rows->Add(row);
                        i++;
                        current = current->link;
                    }

                    //Отображаем таблицу в визуальном компоненте
                    dataGridViewForList->DataSource = table;

                    // Удаляем временный список
                    tmp->deleteTable();
                    delete tmp;
                    head = ML->getList();
                    current = head;
                }
            }

            // После выполнения любого из запросов прячем связанные с ним элементы панели для
индивидуальных запросов
            specialRequestsPanel->Enabled = false;
            specialRequestsPanel->Visible = false;

            // Так же выводим на форму прочие элементы
            dataGridViewForList->Visible = true;

            inputList->Enabled = true;
            outputList->Enabled = true;
            readFile->Enabled = true;
            rewriteFile->Enabled = true;

            specialRequest1->Enabled = true;
            specialRequest2->Enabled = true;
            specialRequest3->Enabled = true;
            specialRequest4->Enabled = true;

            addToList->Enabled = true;
            deleteElement->Enabled = true;
            editElement->Enabled = true;

            // Переходим вв режим ожидания команд
            act = Action::nul;

            // Очищаем текстовые поля
            textBox1->Clear();
            textBox2->Clear();

            return System::Void();
```

```
}

// Обработчики нажатий клавиш в текстовых полях, для фильтрации запрещённых символов
	// Поле названия фильма
System::Void CourseWork::MyForm::fName_KeyPress(System::Object ^ sender,
System::Windows::Forms::KeyPressEventArgs ^ e)
{
	static int wrongClick = 1;
	Char c = e->KeyChar;
	if (c != 8 && c != ' ' && c != ':' && c != '.' && c != ',' && c != '-' && c != '!' &&
c != '?' && !Char::IsLetter(c) && !Char::IsControl(c) && !Char::IsDigit(c))
	{
		e->Handled = true;
		wrongClick++;
		if (wrongClick % 5 == 1)
			MessageBox::Show("Недопустимые символы! Разрешены только буквы, цифры и
знаки препинания.");
	}

	return System::Void();
}
```