

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение   
высшего образования

**«Дальневосточный федеральный университет»**

(ДВФУ)

**ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**

**Департамент программной инженерии и искусственного интеллекта**

ЮГАЛДИНА ЮЛИЯ КОНСТАНТИНОВНА

РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ «КИНОПОИСК»

**КУРСОВОЙ ПРОЕКТ**

по дисциплине «Фундаментальные структуры данных и алгоритмы»

по образовательной программе подготовки бакалавров по направлению

09.03.04 - Программная инженерия

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |  |  | Студентка гр. Б9120-09.03.04прогин | | |
|  |  |  | | Ю.К.Югалдина |
|  | | | | | | |  |  | (подпись) | |  |
|  | | | | | | | | | | | |
| Защищен с оценкой | | | | | | |  |  | Руководитель | | |
|  | | | | | | |  |  | ст. преподаватель | | |
|  | | | |  |  | |  |  | ученая степень, должность |  | О.А. Крестникова |
| (подпись) | | | |  | (И.О. Фамилия) | |  |  | (подпись) |  | (И.О. Фамилия) |
| « |  | » |  | | | 2022 г. |  |  |  | | |
|  | | | | | | | | | | | |

г. Владивосток

2022

# Оглавление

[Оглавление 2](#_Toc116939918)

[Введение 3](#_Toc116939919)

[1 Анализ предметной области 4](#_Toc116939920)

[1.1 Объекты предметной области 4](#_Toc116939921)

[1.2 Законы ПО 6](#_Toc116939922)

[1.3 Постановка задачи «Формирование отчета «Фильмография»» 6](#_Toc116939923)

[2 Теоретическая часть 7](#_Toc116939924)

[2.1 Хеш-таблица 7](#_Toc116939925)

[2.1.1 Хеш-функция 7](#_Toc116939926)

[2.1.2 Разрешение коллизий методом открытой адресации 9](#_Toc116939927)

[2.2 АВЛ-дерево 10](#_Toc116939928)

[2.2.1 Элемент дерева – двусвязный кольцевой список 12](#_Toc116939929)

[3 Требования к информационной системе 14](#_Toc116939930)

[3.1 Требования к данным 14](#_Toc116939931)

[3.1.1 Требования к входным данным 14](#_Toc116939932)

[3.1.2 Требования к выходным данным 15](#_Toc116939933)

[3.2 Функциональные требования 15](#_Toc116939934)

[4 Реализация 18](#_Toc116939935)

[4.1 Спецификация структур данных 18](#_Toc116939936)

[4.2 Описание среды разработки 29](#_Toc116939937)

[4.3 Руководство пользователя 30](#_Toc116939938)

[4.3.1 Работа со справочником «Фильмы» 30](#_Toc116939939)

[4.3.2 Работа со справочником «Актеры» 36](#_Toc116939940)

[4.3.3 Формирование отчета 43](#_Toc116939941)

[4.4 Тестирование 47](#_Toc116939942)

[Заключение 56](#_Toc116939943)

[Список литературы 57](#_Toc116939944)

# Введение

В мире на данный момент существует миллионы разных фильмов. Сейчас просмотр фильма не является чем-то особенным, для этого не обязательно идти в кино и стоять в очереди за билетом. Любой фильм можно найти в интернете на разных онлайн платформах. Но как же выбрать что посмотреть этим вечером? Из-за огромного количества фильмов выбор становится очень сложны. Данный проект направлен на облегчение выбора фильма. В рамках курсового проекта будет рассмотрена информационная система для работы со справочниками фильмов и актеров.

Целью курсового проекта является: разработка информационной системы «Кинопоиск».

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Провести анализ предметной области «Кинопоиск».

2. Изучить теоретические основы методов построения справочников.

3. Определить требования к информационной системе.

4. Реализовать и провести тестирование.

# 1 Анализ предметной области

Требуется разработать информационную систему для предметной области (ПО) «Кинопоиск».

Система должна решать следующие задачи:

1. хранить, позволять просматривать, добавлять и удалять информацию о фильмах (название, жанр, год выхода) и об актерах (ФИО, название фильма, дата рождения);
2. позволять искать информацию о фильмографии по заданному актеру и годам выхода фильмов;
3. формировать список фильмов по названию, жанру, году выпуска.

1.1 Объекты предметной области

Исходя из задач, которые должна решать информационная система, выделены объекты ПО, представленные в Таблице 1.

Таблица 1 – Объекты ПО

|  |  |
| --- | --- |
| Тип и название объекта | Описание объекта |
| Справочник «Фильмы» | Хранит информацию по каждому фильму |
| Справочник «Актеры» | Хранит информацию о каждом актере |
| Отчет «Фильмография» | Хранит информацию по фильмам. Формируется по определённому актеру и выпуску фильма в прокат. |

Каждый фильм характеризуется следующими параметрами: название, жанр, год.

**Название** – строка, содержащая любые символы и не превышает длиной 128 символов.

**Жанр** – строка, содержащая символы русского алфавита и знаки препинания, первое слово начинается с большой буквы.

**Год** – целое число от 1946 до 2022 включительно.

Пример справочника «Фильмы» представлен в таблице 2.

Таблица 2 – Справочник «Фильмы»

|  |  |  |
| --- | --- | --- |
| **Название** | **Жанр** | **Год** |
| Гарри Поттер и орден феникса | Фэнтези | 2007 |
| Полуночное солнце | Драма | 2018 |
| Иллюзия обмана 2 | Триллер | 2016 |
| Красавица и чудовище | Мюзикл | 2017 |
| Алиса в стране чудес | Фэнтези | 2010 |
| Алиса в Зазеркалье | Фэнтези | 2016 |
| Мрачные тени | Фэнтези | 2012 |
| Сонная лощина | Ужасы | 1999 |
| 8 подруг Оушена | Комедия | 2018 |
| Дьявол носит Prada | Комедия | 2006 |

Каждый актер характеризуется следующими параметрами: ФИО, название фильма, год рождения.

**ФИО** – аббревиатура, расшифровывается как: фамилия, имя, отчество. Фамилия, имя и отчество являются словами, состоящими из букв русского алфавита, первая буква каждого слова – заглавная, остальные – строчные. Слова разделены одним пробелом.

**Название** **фильма** – строка, содержащая любые символы и не превышает длиной 128 символов.

**Дата** **рождения** – состоит из трех чисел, разделенных точкой. Первое (день) – целое число от 1 до 31, или от 1 до 30, или от 1 до 28 (в зависимости от месяца), второе (месяц) – целое число от 1 до 12, третье (год) – целое число от 1 до 9999.

Пример справочника «Актеры» представлен в таблице 3.

Таблица 3 – Справочник «Актеры»

|  |  |  |
| --- | --- | --- |
| **ФИО** | **Название фильма** | **Дата рождения** |
| Энн Жаклин Хэтэуэй | Дьявол носит Prada | 12.11.1982 |
| Энн Жаклин Хэтэуэй | 8 подруг Оушена | 12.11.1982 |
| Энн Жаклин Хэтэуэй | Алиса в стране чудес | 12.11.1982 |
| Белла Эйвери Торн | Полуночное солнце | 08.10.1997 |
| Хелена Бонем Картер | Мрачные тени | 26.05.1966 |
| Хелена Бонем Картер | Сонная лощина | 26.05.1966 |
| Хелена Бонем Картер | Алиса в стране чудес | 26.05.1966 |
| Джони Кристофер Депп | Алиса в Зазеркалье | 09.06.1963 |
| Сандра Аннетт Буллок | 8 подруг Оушена | 26.06.1964 |
| Алан Сидни Патрик Рикман | Гарри Поттер и орден феникса | 21.02.1946 |

Пример отчёта «Фильмография» представлен в таблице 4.

Таблица 4 – Пример отчета «Фильмография»

|  |  |  |
| --- | --- | --- |
| **Название** | **Жанр** | **Год** |
| Гарри Поттер и орден феникса | Фэнтези | 2007 |
| Полуночное солнце | Драма | 2018 |
| Иллюзия обмана 2 | Триллер | 2016 |
| Красавица и чудовище | Мюзикл | 2017 |
| Алиса в стране чудес | Фэнтези | 2010 |
| Алиса в Зазеркалье | Фэнтези | 2016 |
| Мрачные тени | Фэнтези | 2012 |
| Сонная лощина | Ужасы | 1999 |
| 8 подруг Оушена | Комедия | 2018 |
| Дьявол носит Prada | Комедия | 2006 |

1.2 Законы ПО

1. у одного актера может быть несколько фильмов;
2. у каждого фильма может быть несколько актеров;
3. фильм не обязательно должен иметь актеров;
4. не может быть актеров у несуществующего фильма;
5. название фильма уникально.

1.3 Постановка задачи «Формирование отчета «Фильмография»»

Входные данные: Справочник «Фильмы», Справочник «Актеры», ФИО актера, нижняя граница диапазона, верхняя граница диапазона .

Выходные данные: Отчет «Фильмография».

Связь:

Отчет «Фильмография».Название = Справочник «Фильмы».Название

Отчет «Фильмография».Жанр = Справочник «Фильмы».Жанр

Отчет «Фильмография».Год = Справочник «Фильмы».Год

Если существует актер в Справочнике «Актеры».ФИО = ФИО актера && Справочник «Актеры».Название фильма = Справочник «Фильмы».Название && Справочник «Фмльмы».Год ≥ нижняя граница диапозона && Справочник «Фмльмы».Год ≤ верхняя граница диапазона.

# 2 Теоретическая часть

Существует множество различных структур данных, таких как массив, список, бинарное дерево, хеш-таблица, стек, очередь, и т. д. Для ускорения поиска используются, например, сбалансированные бинарные деревья поиска (АВЛ-деревья, красно-черные деревья, B-деревья и т.д.) и хеш-таблицы.

Для работы со справочником «Фильмы», а именно с добавлением в него записей, требуется проверка на уникальность добавляемых данных. Поскольку ключ уникален, то для этого поиска используется хеш-таблица. Из-за большого по размеру ключа, была выбрана хеш-функция свёртки по два. Хеш-таблица является статической, поскольку количество записей в справочнике ограничено (не превышает 1000 записей). Разрешение коллизий происходит методом открытой адресации, так как он занимает меньше памяти, относительно метода цепочек. [1].

Для хранения, просмотра, добавления, удаления и поиска актеров было выбрано АВЛ дерево, где ключ ФИО актера, исходя из того, что у Актера может быть несколько фильмов, то требуется обеспечить быстрый поиск, добавление и удаление.

2.1 Хеш-таблица

Хеш-таблица представляет собой эффективную структуру данных для реализации словарей. Хотя на поиск элемента в хеш-таблице может в наихудшем случае потребоваться O(n), на практике хеширование исключительно эффективно. При вполне обоснованных допущениях среднее время поиска элемента в хеш-таблице составляет O(1). [1].

Хеш-таблица – это массив T [0.. m-1], в котором номер ячейки для ключа keyмножество уникальных ключей, вычисляется с использованием хеш-функции [1].

2.1.1 Хеш-функция

Хеш-функция принимает в качестве аргумента ключ из множества U и возвращает соответствующий ему номер ячейки таблицы T [0..m-1]. Другими словами, хеш-функция отображает совокупность ключей U на множество ячеек хеш-таблицы T [0..m-1]: [1].

Рассмотрим пример построения хеш-функции на примере метода свёртки по два. Построение хеш-функции этим методом состоит в отображении ключа k на одну из 𝑚 ячеек путем получения остатка от деления 𝑘 на 𝑚, где 𝑘 - сумма всех цифр в ключе. [1].

В рамках курсового проекта ключами в хеш-таблице является строковое поле – название фильма (key). Число key является суммой чисел, являющимися кодами символов в кодировке UTF-8.

Предположим, что размер хеш-таблицы составляет 10. Вычислим значение хеш-функции для записей из модели предметной области. Пример:

1. key (Гарри Поттер и орден феникса) = 1043 + 1072 + 1088 + 1088 + 1080 + 32 + 1055 + 1086 + 1090 + 1090 + 1077 + 1088 + 32 + 1080 + 32 + 1086 + 1088 + 1076 + 1077 + 1085 + 32 + 1092 + 1077 + 1085 + 1080 + 1082 + 1089 + 1072 = 26054

h(26054) = (2 + 60 + 54) mod 10 = 116 mod 10 = 6

1. key (Полуночное солнце) = 1055 + 1086 + 1083 + 1091 + 1085 + 1086 + 1095 + 1085 + 1086 + 1077 + 32 + 1089 + 1086 + 1083 + 1085 + 1094 + 1077 = 17375

h(17375) = (1 + 73 + 75) mod 10 = 149 mod 10 = 9

1. key (Иллюзия обмана 2) = 1048 + 1083 + 1083 + 1102 + 1079 + 1080 + 1103 + 32 + 1086 + 1073 + 1084 + 1072 + 1085 + 1072 + 32 + 50 = 14164

h(14164) = (1 + 41 + 64) mod 10 = 106 mod 10 = 6

1. key (Красавица и чудовище) = 19511

h(19511) = 7

1. key (Алиса в стране чудес) = 18463

h(18463) = 8

1. key (Алиса в Зазеркалье) = 17279

h(17279) = 2

1. key (Мрачные тени) = 11932

h(11932) = 2

1. key (Сонная лощина) = 13023

h(13023) = 4

1. key (8 подруг Оушена) = 13098

h(13098) = 9

1. key (Дьявол носит Prada) = 12472

h(12472) = 7

Как видно из примера, два ключа могут быть хешированы в одну и ту же ячейку, например, для ключей (Мрачные тени) и (Алиса в Зазеркалье). Такая ситуация называется коллизией.

Существует несколько эффективных методов разрешения коллизий: например, метод открытой адресации, метод цепочек, пакетирование. В рамках курсового проекта для разрешения коллизий в хеш-таблице будет рассмотрен метод открытой адресации, а именно – линейное пробирование.

2.1.2 Разрешение коллизий методом открытой адресации

Простейший метод открытой адресации называется линейным зондированием (или линейным пробированием): при наличии конфликта (когда хеширование выполняется в место таблицы, которое уже занято элементом с ключом, не совпадающим с ключом поиска) мы просто проверяем следующую позицию в таблице. Обычно подобную проверку (определяющую, содержит ли данная позиция таблицы элемент с ключом, равным ключу поиска) называют зондированием (пробированием) [2].

Линейное зондирование характеризуется выявлением одного из трех возможных исходов зондирования: если позиция таблицы содержит элемент, ключ которого совпадает с искомым, имеет место попадание при поиске; в противном случае (если позиция таблицы содержит элемент, ключ которого не совпадает с искомым) мы просто зондируем позицию таблицы со следующим по величине индексом, продолжая этот процесс (возвращаясь к началу таблицы при достижении ее конца) до тех пор, пока не будет найден искомый ключ или пустая позиция таблицы [2].

На рисунке 1 представлено разрешение коллизий в виде линейного пробирования.

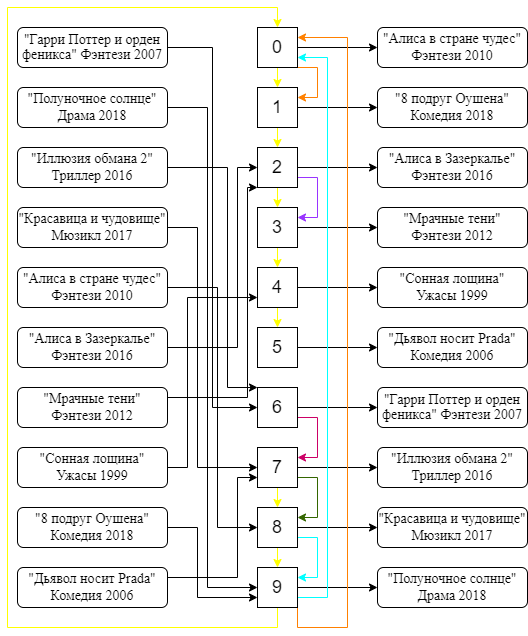


Рисунок 1 - Метод открытой адресации.

2.2 АВЛ-дерево

АВЛ-дерево — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1 [3].

Показатель сбалансированности в дальнейшем будем интерпретировать как разность между высотой левого и правого поддерева. Непосредственно при вставке (листу) присваивается нулевой баланс. Процесс включения вершины состоит из трех частей (данный процесс описан Никлаусом Виртом в «Алгоритмы и структуры данных»):

Алгоритм добавления вершины:

1. Прохода по пути поиска, пока не убедимся, что ключа в дереве нет;
2. Вставить новый узел и определить получающийся баланс;
3. Вернуться по пути поиска и проверить баланс в каждом узле. Если нужно выполнить балансировку. [3]

Будем возвращать в качестве результата функции, уменьшилась высота дерева или нет. Предположим, что процесс из левой ветви возвращается к родителю (рекурсия идет назад), тогда возможны три случая: {hl — высота левого поддерева, hr — высота правого поддерева} Включение вершины в левое поддерево приведет к:

1. hl = hr: высота левого и правого поддерева становится неравной, но критерий сбалансированности не нарушается.
2. hl < hr: высота левого и правого поддерева становится равной, то есть баланс только улучшается.
3. hl > hr: критерий сбалансированности нарушается, дерево нужно перестраивать.

В третьей ситуации требуется определить балансировку левого поддерева. Если левое поддерево этой вершины выше правого, то требуется большое правое вращение, иначе хватит малого правого. Аналогичные (симметричные) рассуждения можно привести и для включения в правое поддерево [3].

Алгоритм удаления вершины:

Вершина — лист, удалим её и вызовем балансировку всех её предков в порядке от родителя к корню [3].

Докажем, что данный алгоритм сохраняет балансировку. Для этого докажем по индукции по высоте дерева, что после удаления некоторой вершины из дерева и последующей балансировки высота дерева уменьшается не более, чем на 1. База индукции: для листа очевидно, верно. Шаг индукции: либо условие сбалансированности в корне (после удаления корень может изменится) не нарушилось, тогда высота данного дерева не изменилась, либо уменьшилось строго меньшее из поддеревьев => высота до балансировки не изменилась => после уменьшится не более чем на 1.[3]

Очевидно, что в результате указанных действий процедура удаления вызывается не более 3 раз, так как у вершины, удаляемой по второму вызову, нет одного из поддеревьев. Но поиск ближайшего каждый раз требует O(N) операций. Становится очевидной возможность оптимизации: поиск ближайшей вершины может быть выполнен по краю поддерева, что сокращает сложность до O(log(N)) [1].

Расстановка балансов при удалении:

Удаляемая вершина — лист, она удаляется, и обратный обход дерева происходит от родителя удалённого листа.

Если при переходе к родителю пришли слева — баланс увеличивается на 1, если же пришли справа — уменьшается на 1.

Это делается до тех пор, пока при изменении баланса он не станет равным −1 или 1: в данном случае такое изменение баланса будет гласить о неизменной дельта-высоте поддеревьев. Повороты происходят по тем же правилам, что и при вставке [1].

На рисунке 2 представлено АВЛ-дерево (см. 2.2.1).

2.2.1 Элемент дерева – двусвязный кольцевой список

Линейные списки являются чрезвычайно гибкой структурой, так как их легко сделать большими или меньшими, и их элементы доступны для вставки или удаления в любой позиции списка. Списки также можно объединять или разбивать на меньшие списки [3].

Элементы списка могут быть хаотично распределены по памяти. Из-за этого теряется возможность быстро получить элемент по индексу, а также не представляется возможным быстро скопировать весь список, однако, есть возможность вставки элементов за линейное время в любое место [3].

В качестве списка в рамках курсового проекта был выбран двусвязный кольцевой список.

Каждый узел двунаправленного (двусвязного) кольцевого списка содержит два поля указателей — на следующий и на предыдущий узлы. Указатель на предыдущий узел корня списка содержит адрес последнего узла. Указатель на следующий узел последнего узла содержит адрес корня списка. При добавлении нового элемента, он добавляется в конец. Удаление элемента происходит по значению с последующим переопределением связей [4].



Рисунок 2 - Визуальное представление АВЛ-дерева

# 3 Требования к информационной системе

В данной главе описываются требования к информационной системе «Кинопоиск», а именно: требования к входным данным системы, требования к её выходным данным и требования к её функционалу.

3.1 Требования к данным

3.1.1 Требования к входным данным

Основываясь на анализе ПО, входными данными является:

* текстовый файл film.txt, каждая строка файла содержит информацию об объекте «Фильмы», а именно название, жанр и год выхода, разделенный символом «|»;

Пример текстового файла:

Гарри Поттер и орден феникса | Фэнтези | 2007

Полуночное солнце | Драма | 2018

Иллюзия обмана 2 | Триллер | 2016

Красавица и чудовище | Мюзикл | 2017

Алиса в стране чудес | Фэнтези | 2010

Алиса в Зазеркалье | Фэнтези | 2016

Мрачные тени | Фэнтези | 2012

Сонная лощина | Ужасы | 1999

8 подруг Оушена | Комедия | 2018

Дьявол носит Prada | Комедия | 2006

* текстовый файл actor.txt, каждая строка файла содержит информацию об объекте «Актеры», а именно ФИО, название фильма, дата рождения, разделенный символом «|»;

Пример текстового файла:

Энн | Жаклин | Хэтэуэй | Дьявол носит Prada | 12.11.1982

Энн | Жаклин | Хэтэуэй | 8 подруг Оушена | 12.11.1982

Энн | Жаклин | Хэтэуэй | Алиса в стране чудес | 12.11.1982

Белла | Эйвери | Торн | Полуночное солнце | 08.10.1997

Хелена | Бонем | Картер | Мрачные тени | 26.05.1966

Хелена | Бонем | Картер | Сонная лощина | 26.05.1966

Хелена | Бонем | Картер | Алиса в стране чудес | 26.05.1966

Джони | Кристофер | Депп | Алиса в Зазеркалье | 09.06.1963

Сандра | Аннетт | Буллок | 8 подруг Оушена | 26.06.1964

Алан | Сидни Патрик | Рикман | Гарри Поттер и орден феникса | 21.02.1946

* ФИО;
* Дата рождения;
* Название фильма;
* Жанр;
* Год выхода;

3.1.2 Требования к выходным данным

Выходными данными являются:

* текстовый файл film.txt, каждая строка файла содержит информацию об объекте «Фильмы», а именно название, жанр и год выхода, разделенный символом «|»;
* текстовый файл actor.txt, каждая строка файла содержит информацию об объекте «Актеры», а именно ФИО, название фильма, дата рождения, разделенный символом «|»;
* текстовый файл report.txt, каждая строка файла содержит информацию из отчета «Фильмография», а именно название, жанр и год выхода, разделенный символом «|»;
* «Данные введены некорректно»
* «Хеш-таблица переполнена»
* «Этот элемент уже есть в таблице»
* «Такого фильма не существует в списке «Фильмы. Добавьте фильм.»

3.2 Функциональные требования

Информационная система, должна позволять:

* считывать данные справочников из текстовых файлов (см п 3.1.1);
* просматривать всю информацию о фильмах из справочника “Фильмы”;
* добавлять информацию в справочник «Фильмы» по значениям: название, жанр, год выхода. При добавлении фильма в заполненную хеш-таблицу программа должна выводить соответствующее сообщение;
* проверять добавляемые значения в справочник «Фильмы» на корректность, а именно: год выхода – целое число от 1946 до 2022 включительно, жанр – строка, содержащая символы русского алфавита и знаки препинания, первое слово начинается с большой буквы;
* просматривать всю информацию об актерах из справочника “Актеры”;
* добавлять информацию в справочник «Актеры» по значениям: ФИО, название фильма, дата рождения;
* проверять добавляемые значения в справочник «Актеры» на корректность, а именно: ФИО – строка содержащая символы русского алфавита, фамилия, имя и отчество являются словами, первая буква каждого слова – заглавная, остальные – строчные;
* проверять на уникальность фильм, введенный пользователем, при добавлении в справочник «Фильмы», если фильм уже существует в таблице, программа должна выводить соответствующее сообщение;
* соблюдать целостность между справочниками «Фильмы» и «Актеры» при добавлении в справочнике «Актеры», т. е. если название фильма есть в справочнике «Фильмы», то добавляем в справочник, иначе программа должна выводить соответствующее сообщение;
* удалять информацию из справочника «Фильмы» по значениям: название, год выхода;
* удалять информацию из справочника «Актеры» по значениям: ФИО;
* удалять информацию из справочника «Актеры» по значениям: ФИО, название фильма;
* соблюдать целостность между справочниками «Фильмы» и «Актеры» при удалении из справочника «Фильмы», т. е. если у фильма нет связных записей в справочнике «Актеры», то удалить из справочника, иначе удалить соответствующие данные из справочника «Актеры» и удалить фильм;
* находить информацию в справочниках «Актеры» и «Фильмы» по значениям: ФИО актера, и диапазону лет выхода фильма, год (от) должен быть меньше или равен (до);
* сохранять все внесённые изменения в справочники «Фильмы» и «Актеры» в файлы (см. п. 3.1.2);
* создавать отчет «Фильмография» по заданному ФИО актера, и диапазону лет выхода фильма;
* сохранять отчет «Фильмография» в заданный файл (см п.3.1.2);

# 4 Реализация

В данной главе содержится спецификация реализованных структур данных, описание графического интерфейса, а также приведены результаты тестирования структур данных.

4.1 Спецификация структур данных

Класс **Film** - класс, описывающий фильм.

Поля:

* string name – название фильма;
* string genre– жанры фильма;
* int date – год выхода фильма;
* int status – статус, описывающий состояние ячейки в хеш-таблице (empty = 0, filled = 1, deleted = 2);

Методы:

* Film() – конструктор класса, создает экземпляр класса, с пустыми данными;

Входные данные: отсутствуют;

Формальные параметры: отсутствуют;

Выходные данные: объект класса Film.

* Film(string nf, string g, int d) – конструктор класса, создает экземпляр класса, с заданными параметрами, статус равен нулю;

Входные данные: отсутствуют;

Формальные параметры: string nf – название фильма, string g - жанр, int d – год выхода;

Выходные данные: объект класса Film.

* void WriteF() – функция выводящая на экран поля экземпляра класса.

Входные данные: объект класса Film;

Формальные параметры: отсутствуют;

Выходные данные: вывод объекта класса Film.

Класс **Actor** - класс, описывающий актера.

Поля:

* string f – фамилия актера;
* string i – имя актера;
* string o – отчество или второе имя актера;
* string name\_f – название фильма где снимался актер;
* int day – день рождения актера;
* int month – месяц рождения актера;
* int years – год рождения актера;

Методы:

* Actor() – конструктор класса, создает экземпляр класса, с пустыми данными;

Входные данные: отсутствуют;

Формальные параметры: отсутствуют;

Выходные данные: объект класса Actor.

* Actor(string im, string ot, string fam, string nf, int d, int m, int y) – конструктор класса, создает экземпляр класса, с заданными параметрами;

Входные данные: отсутствуют;

Формальные параметры: string im – имя актера, string ot – отчество или второе имя актера, string fam – фамилия актера, string nf – название фильма в котором он участвовал, int d – день рождения, int m – месяц рождения, int y – год рождения;

Выходные данные: объект класса Actor.

* Actor(string fam, string im, string ot) – конструктор класса, создает экземпляр класса, с заданными параметрами, название фильма и дата рождения пустые;

Входные данные: отсутствуют;

Формальные параметры: string im – имя актера, string ot – отчество или второе имя актера, string fam – фамилия актера;

Выходные данные: объект класса Actor.

* Actor(string fam, string im, string ot, string n) – конструктор класса, создает экземпляр класса, с заданными параметрами, дата рождения пустая;

Входные данные: отсутствуют;

Формальные параметры: string im – имя актера, string ot – отчество или второе имя актера, string fam – фамилия актера, string nf – название фильма в котором он участвовал;

Выходные данные: объект класса Actor.

Класс **Spisok** - класс, описывающий двусвязный кольцевой список со вставкой в конец.

Поля:

* Actor act – значение элемента списка;
* Spisok next – указатель на следующий элемент;
* Spisok prev – указатель на предыдущий элемент;
* Spisok start – указать на начало списка;

Методы:

* Spisok() – конструктор списка;

Входные данные: отсутствуют;

Формальные параметры: отсутствуют;

Выходные данные: инициализированный список.

* void Add(Actor fil, AVL.Node tree) – добавляет элемент Actor в список в начало;

Входные данные: двусвязный кольцевой список, элемент, который добавляется, узел дерева;

Формальные параметры: переменная Actor fil – элемент, который добавляется в список, AVL.Node tree – узел дерева, к которому прикреплен список;

Выходные данные: двусвязный кольцевой список со вставленным элементом fil.

* Del\_Ver(AVL.Node tree) – удаляет узел дерева и заменяет его на начало списка;

Входные данные: двусвязный кольцевой список, узел дерева;

Формальные параметры: AVL.Node tree – узел дерева, к которому прикреплен список;

Выходные данные: новый узел дерева, на месте которого первый элемент списка.

* void Del\_F(string f, AVL.Node tree) – удаляет элемент из списка;

Входные данные: двусвязный кольцевой список, название фильма, в котором снимался актер, узел дерева;

Формальные параметры: string f - название фильма, AVL.Node tree – узел дерева, к которому прикреплен список;

Выходные данные: двусвязный список без элемента в поле которого был фильм под название f.

* void Find(string[] mas) – записывает данные из списка, в массив строк;

Входные данные: двусвязный кольцевой список, массив строк;

Формальные параметры: string[] mas – массив строк, в который записываются фильмы;

Выходные данные: массив строк, заполненный фильмами, в которых снимался актер.

* void Find\_A(Actor[] mas) – записывает данные из списка, в массив экземпляров класса Actor;

Входные данные: двусвязный кольцевой список, массив экземпляров класса Actor;

Формальные параметры: Actor[] mas – массив экземпляров класса Actor, в который записываются данные каждого узла списка;

Выходные данные: массив экземпляров класса Actor.

* void view\_begin() – записывает данные списка в массив;

Входные данные: двусвязный кольцевой список, массив экземпляров класса Actor ;

Формальные параметры: отсутствуют;

Выходные данные: заполненный массив экземпляров класса Actor.

* void view\_begin\_file() – записывает данные из списка в файл;

Входные данные: двусвязный кольцевой список;

Формальные параметры: отсутствуют;

Выходные данные: заполненный массив экземпляров класса Actor.

* void Clear(AVL.Node tree) – удаляет данные из списка;

Входные данные: двусвязный кольцевой список, узел дерева;

Формальные параметры: AVL.Node tree – узел дерева;

Выходные данные: очищенный двусвязный список.

Класс **AVL** - класс, АВЛ-дерево.

Поля:

Поля:

* Класс **Node** – класс, описывающий узел дерева;
* Поля:
  + Actor key – значение узла дерева;
  + Node left – указатель на левый потомок узла дерева;
  + Node right – указатель на правый потомок узла дерева;
  + Spisok coun – кольцевой двусвязный список;
* Методы
* Node() – конструктор класса, создает экземпляр класса, с пустыми данными;
* Node(Actor a) – конструктор класса, создает экземпляр класса, с заданными параметрами.
* Node root – корень дерева;
* Actor[] mas – массив экземпляров класса Actor;
* int ind – индекс последнего элемента массива mas;

Методы:

* int Comparison(Actor a, Node tree) – функция сравнения двух полей класса Actor по ФИО актера;

Входные данные: экземпляр класса Actor и узел дерева;

Формальные параметры: Actor a – экземпляр класса Actor, Node tree – узел дерева;

Выходные данные: если экземпляр класса Actor, меньше чем ключ узла дерева, то возвращает -1, если больше то 1, если равны то 0.

* void new\_FA() – записывает данные из дерева в файл;

Входные данные: дерево;

Формальные параметры: отсутствуют;

Выходные данные: заполненный файл.

* void Add(Actor act) – добавляет новый узел в дерево;

Входные данные: дерево, экземпляр класса Actor;

Формальные параметры: Actor act – экземпляр класса Actor ;

Выходные данные: дерево с узлом, ключ которого равен act.

* Node RecursiveInsert(Node t1, Node t2) - функция отвечающая за балансировку дерева при вставке.

Входные данные: дерево, экземпляр класса Actor;

Формальные параметры: Node t1 – узел дерева, Node t2 – узел дерева;

Выходные данные: возвращает текущий корень дерева.

* Node balance\_tree(Node t1) – функция проверяет баланс вершины и определяет нужен ли поворот;

Входные данные: дерево, узел дерева;

Формальные параметры: Node t1 – узел дерева;

Выходные данные: возвращает уже сбалансированную вершину.

* Delete\_F(string name\_f) – удаляет все узлы, в ключе которого есть заданный фильм;

Входные данные: дерево, название фильма;

Формальные параметры: string name\_f – строка, содержащая название фильма;

Выходные данные: изменённое дерево.

* void Delete(Actor a) – удаляет из дерева узел с ФИО определенного актера;

Входные данные: дерево, экземпляр класса Actor;

Формальные параметры: Actor a – экземпляр класса Actor, содержащий ФИО актера, которого нужно удалить;

Выходные данные: изменённое дерево.

* void Delete\_AF(Actor a) - удаляет из дерева узел с заданным ФИО актера и фильма где он снимался;

Входные данные: дерево, экземпляр класса Actor;

Формальные параметры: Actor a – экземпляр класса Actor, содержащий ФИО актера и фильм, который нужно удалить;

Выходные данные: изменённое дерево.

* void Find(Actor a, string[] mas) – ищет актера в дереве по ФИО и записывает в массив фильмы, где он участвовал;

Входные данные: дерево, экземпляр класса Actor, массив строк;

Формальные параметры: Actor a – экземпляр класса Actor, содержащий ФИО актера, string[] mas – массив строк для записи фильмов, где снимался определенный актер;

Выходные данные: заполненный массив строк.

* void Find\_A(Actor a, Actor [] mas) – ищет актера в дереве по ФИО и записывает в массив все данные о нем;

Входные данные: дерево, экземпляр класса Actor, массив экземпляров класса Actor;

Формальные параметры: Actor a – экземпляр класса Actor, содержащий ФИО актера, Actor[] mas – массив экземпляров класса Actor для записи данных об актере;

Выходные данные: заполненный массив экземпляров класса Actor.

* void DisplayTree() - функция обходит дерево и записывает узлы в массив;

Входные данные: дерево, массив экземпляров класса Actor;

Формальные параметры: Actor a – экземпляр класса Actor, содержащий ФИО актера, string[] mas – массив строк для записи фильмов, где снимался определенный актер;

Выходные данные: заполненный массив строк.

* int max(int l, int r) – вычисляет какое поддерево больше;

Входные данные: высота правого поддерева, высота левого поддерева;

Формальные параметры: int l – высота левого поддерева, int r – высота правого поддерева;

Выходные данные: возвращает большее поддерево.

* int getHeight(Node current) – возвращает наибольшую высоту узла.

Входные данные: узел дерева;

Формальные параметры: Node current – узел дерева;

Выходные данные: возвращает наибольшую высоту узла.

* int balance\_factor(Node current) – вычисляет баланс вершины;

Входные данные: узел дерева;

Формальные параметры: Node current – узел дерева;

Выходные данные: возвращает баланс вершины.

* Node RotateRR(Node parent) – выполняет правый поворот в заданном узле дерева;

Входные данные: узел, в котором происходит поворот;

Формальные параметры: переменная Node parent – узел, в котором происходит балансировка;

Выходные данные: сбалансированный узел.

* Node RotateLL(Node parent) – выполняет левый поворот в заданном узле дерева;

Входные данные: узел, в котором происходит поворот;

Формальные параметры: переменная Node parent – узел, в котором происходит балансировка;

Выходные данные: сбалансированный узел.

* Node RotateLR(Node parent) – выполняет лево-правый поворот в заданном узле дерева;

Входные данные: узел, в котором происходит поворот;

Формальные параметры: переменная Node parent – узел, в котором происходит балансировка;

Выходные данные: сбалансированный узел.

* Node RotateRL(Node parent) – выполняет право-левый поворот в заданном узле дерева;

Входные данные: узел, в котором происходит поворот;

Формальные параметры: переменная Node parent – узел, в котором происходит балансировка;

Выходные данные: сбалансированный узел.

Класс **HeshT** - класс, описывающий хеш-таблицу.

Поля:

* int size – размер хеш-таблицы;
* Film[] h – массив всех элементов в таблице;
* int k = 1 – коэффициент, для вычисления вторичной хеш-функции;

Методы:

* HeshT(int N) – конструктор класса, создает хеш-таблицу, с размером N;
* int hashkey(string i) – функция переводящая строку в число;

Входные данные: строка;

Формальные параметры: string i – строка, название фильма;

Выходные данные: возвращает число, равное сумме всех символов в кодировке UTF-8.

Пример:

На вход поступает запись, содержащая информацию о фильме из ПО (см. 1.1), а именно

Название фильма:

1. key (Гарри Поттер и орден феникса) = 1043 + 1072 + 1088 + 1088 + 1080 + 32 + 1055 + 1086 + 1090 + 1090 + 1077 + 1088 + 32 + 1080 + 32 + 1086 + 1088 + 1076 + 1077 + 1085 + 32 + 1092 + 1077 + 1085 + 1080 + 1082 + 1089 + 1072 = 26054
2. key (Полуночное солнце) = 1055 + 1086 + 1083 + 1091 + 1085 + 1086 + 1095 + 1085 + 1086 + 1077 + 32 + 1089 + 1086 + 1083 + 1085 + 1094 + 1077 = 17375
3. key (Иллюзия обмана 2) = 1048 + 1083 + 1083 + 1102 + 1079 + 1080 + 1103 + 32 + 1086 + 1073 + 1084 + 1072 + 1085 + 1072 + 32 + 50 = 14164
4. key (Красавица и чудовище) = 19511
5. key (Алиса в стране чудес) = 18463
6. key (Алиса в Зазеркалье) = 17279
7. key (Мрачные тени) = 11932
8. key (Сонная лощина) = 13023
9. key (8 подруг Оушена) = 13098
10. key (Дьявол носит Prada) = 12472

* int HeshF(int key) – функция вычисляет первичную хеш-функцию;

Входные данные: значение первичной хеш-функции;

Формальные параметры int key – число, являющиеся значением первичной хеш-функции;

Выходные данные: возвращает число – вторичная хеш-функция заданного ключа.

Пример:

Предположим, что размер ХТ равен 10, а на вход поступает запись, содержащая информацию о фильме из ПО (см. 1.1), а именно

Значение ключа:

1. h(26054) = (2 + 60 + 54) mod 10 = 116 mod 10 = 6
2. h(17375) = (1 + 73 + 75) mod 10 = 149 mod 10 = 9
3. h(14164) = (1 + 41 + 64) mod 10 = 106 mod 10 = 6
4. h(19511) = 7
5. h(18463) = 8
6. h(17279) = 2
7. h(11932) = 2
8. h(13023) = 4
9. h(13098) = 9
10. h(12472) = 7

* int HeshF2(int key, int j) - функция вычисляет вторичную хеш-функцию;

Входные данные: ключ для хеш-функции, номер попытки;

Формальные параметры int key – число, являющиеся ключом для вычисления хеш-функции, int j – число, означающие номер попытки;

Выходные данные: возвращает число – вторичная хеш-функция заданного ключа.

Пример:

Предположим, что размер ХТ равен 10, а на вход поступает запись, содержащая информацию о фильме из ПО (см. 1.1), а именно

Значение ключа:

(key + j\*1) mod 10

1. h(6) = (6 + 1\*1) mod 10 = 7 mod 10 = 7
2. h(9) = (9 + 1\*1) mod 10 = 10 mod 10 = 0
3. h(6) = (6 + 1\*1) mod 10 = 7 mod 10 = 7
4. h(7) = 8
5. h(8) = 9
6. h(2) = 3
7. h(2) = 3
8. h(4) = 5
9. h(9) = 0
10. h(7) = 8

* int Collizia(int key, Film i) – функция, которая разрешает коллизию;

Входные данные: значение первой хеш-функции, экземпляр класса Film;

Формальные параметры int key – число, являющиеся ключом для вычисления хеш-функции, Film i – экземпляр класса Film, который нужно добавить в хеш-таблицу;

Выходные данные: индекс, для добавления в хеш-таблицу.

* int DelCollizia(int ind, string name) – функция ищет значение элемента в таблице при удалении, в случае коллизии;

Входные данные: значение первой хеш-функции, название фильма;

Формальные параметры int key – число, являющиеся ключом для вычисления хеш-функции, string name – строка, в которой лежит название фильма;

Выходные данные: положение элемента в таблице.

* void New\_F() – записывает данные из таблицы в файл;

Входные данные: хеш-таблица;

Формальные параметры: отсутствуют;

Выходные данные: заполненный файл.

* int Add(Film i) – добавляет элемент в таблицу;

Входные данные: хеш-таблица, экземпляр класса Film;

Формальные параметры: Film i - экземпляр класса Film, который нужно добавить в таблицу;

Выходные данные: хеш-таблица с добавленным элементом, возвращает положение элемента в таблице.

* int Del(string name, int years) – удаляет элемент из таблицы;

Входные данные: хеш-таблица, название фильма, год выхода;

Формальные параметры: string name – строка, содержащая название фильма, int years – целое число;

Выходные данные: хеш-таблица с удаленным элементом, возвращает индекс, удаленного элемента.

* ind Search\_ (string name) – ищет элемент в таблице по заданным критериям;

Входные данные: хеш-таблица, название фильма;

Формальные параметры: string name – строка, содержащая название фильма;

Выходные данные: возвращает индекс найденного элемента.

* void Search\_File(string name, int y1, int y2) – ищет в таблице элемент по заданным критериям и записывает его в файл;

Входные данные: хеш-таблица, название фильма, диапазон годов;

Формальные параметры: string name – строка, содержащая название фильма, int y1 – целое число, означающие нижнюю границу диапазона, int y2 – целое число, означающие верхнюю границу диапазона;

Выходные данные: заполненный файл.

4.2 Описание среды разработки

Интегрированная среда разработки Visual Studio — это стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений. Помимо стандартного редактора и отладчика, которые есть в большинстве сред IDE, Visual Studio включает в себя компиляторы, средства автозавершения кода, графические конструкторы и многие другие функции для улучшения процесса разработки.

Вот несколько популярных возможностей Visual Studio, которые повышают производительность при разработке программного обеспечения:

* Волнистые линии и быстрые действия

Волнистые линии обозначают ошибки или потенциальные проблемы кода прямо во время ввода. Эти визуальные подсказки помогают немедленно устранить проблемы, не дожидаясь появления ошибок во время сборки или выполнения. Если навести указатель мыши на волнистую линию, на экран будут выведены дополнительные сведения об ошибке. Также в поле слева может отображаться лампочка, указывающая на наличие сведений о быстрых действиях для устранения ошибки.

* IntelliSense

IntelliSense — это набор возможностей, отображающих сведения о коде непосредственно в редакторе и в некоторых случаях автоматически создающих небольшие отрывки кода. По сути, это 28 встроенная в редактор базовая документация, которая избавляет от необходимости искать информацию в других источниках.

Данная среда разработки была выбрана мною, потому что она является одной из самых популярных сред разработки и потому что у меня нет опыта с другими средами.

4.3 Руководство пользователя

**4.3.1 Работа со справочником «Фильмы»**

При запуске приложения Курсовой Проект.exe открывается окно программы для работы со справочником «Фильмы» (рисунок 3).

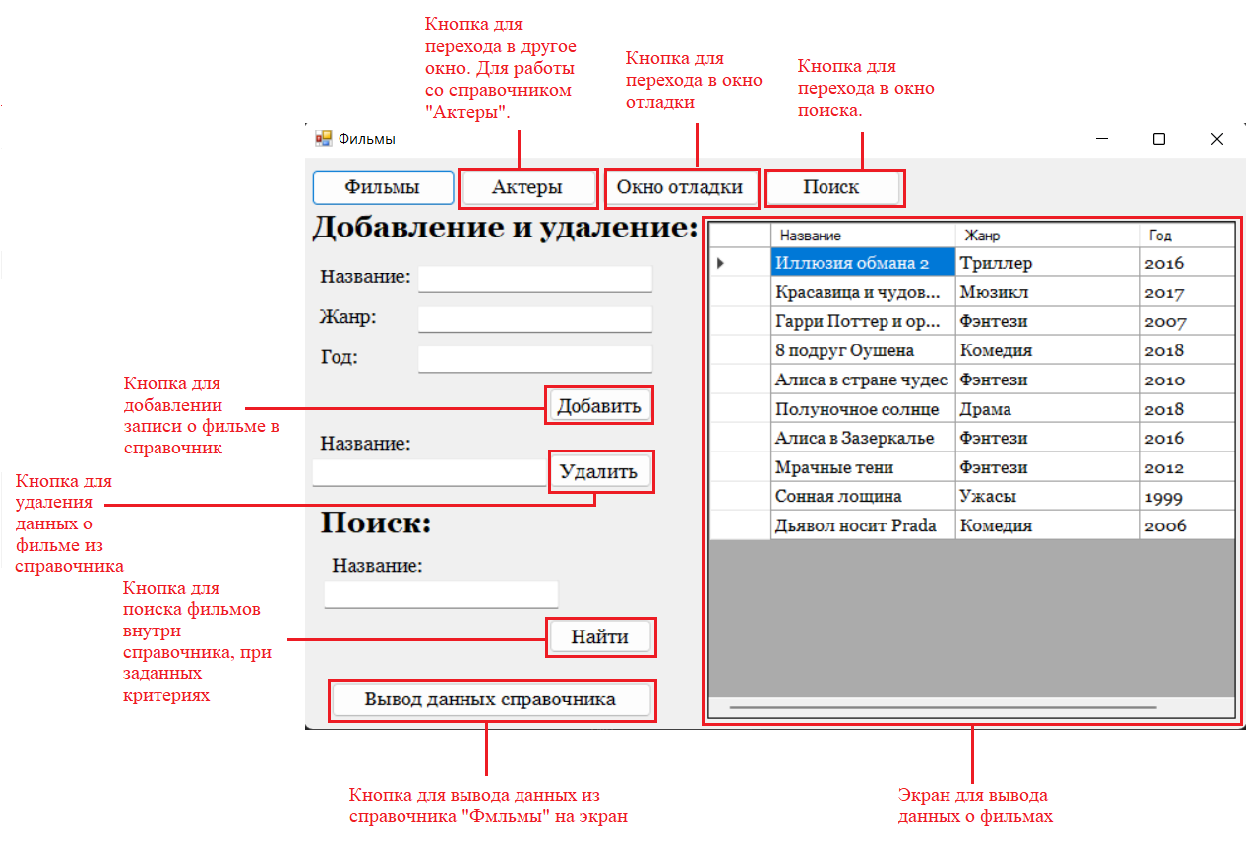


Рисунок 3 - работа со справочником «Фильмы»

**Добавление**

Можно добавлять данные в справочник «Фильмы» собственноручно. Для демонстрации будем добавлять данные в справочник, заполненный записями из ПО (см. 1.1). Для этого необходимо заполнить поля соотвуетствующими данными и нажать на кнопку «Добавить» (рисунок 4.1 – 4.2);

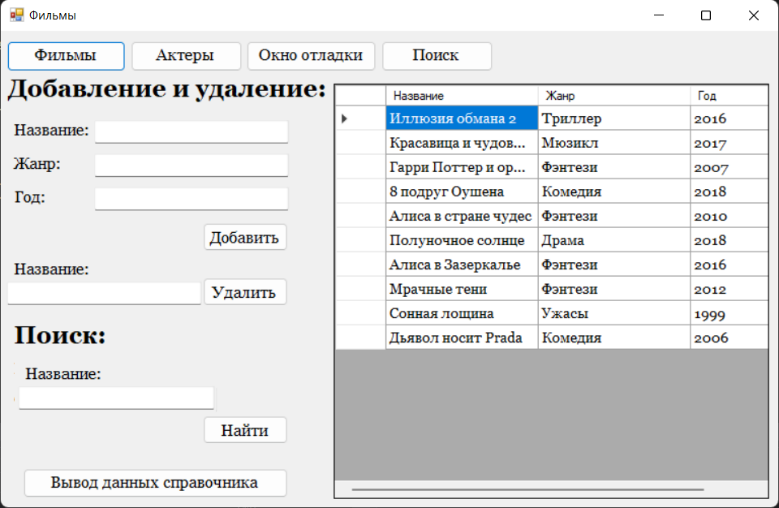
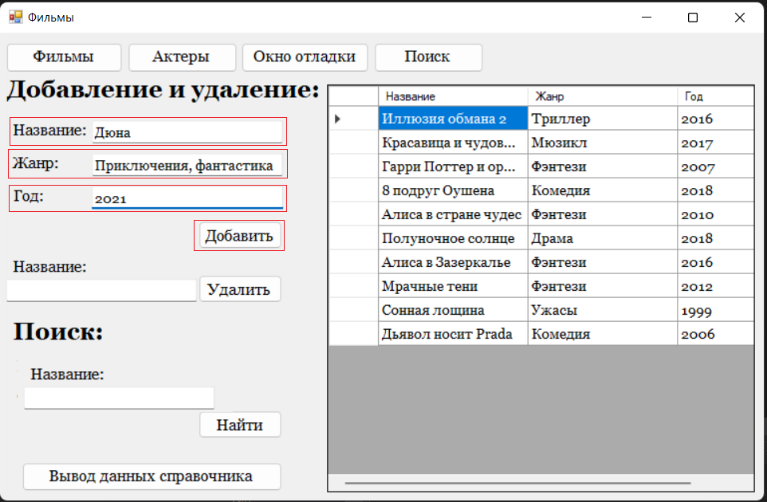


Рисунок 4.2 - Заполнение данных для добавления в справочник

Рисунок 4.1 - окно справочника "Фильмы"

Данные справочника отобразятся в главном окне (рисунок 5).

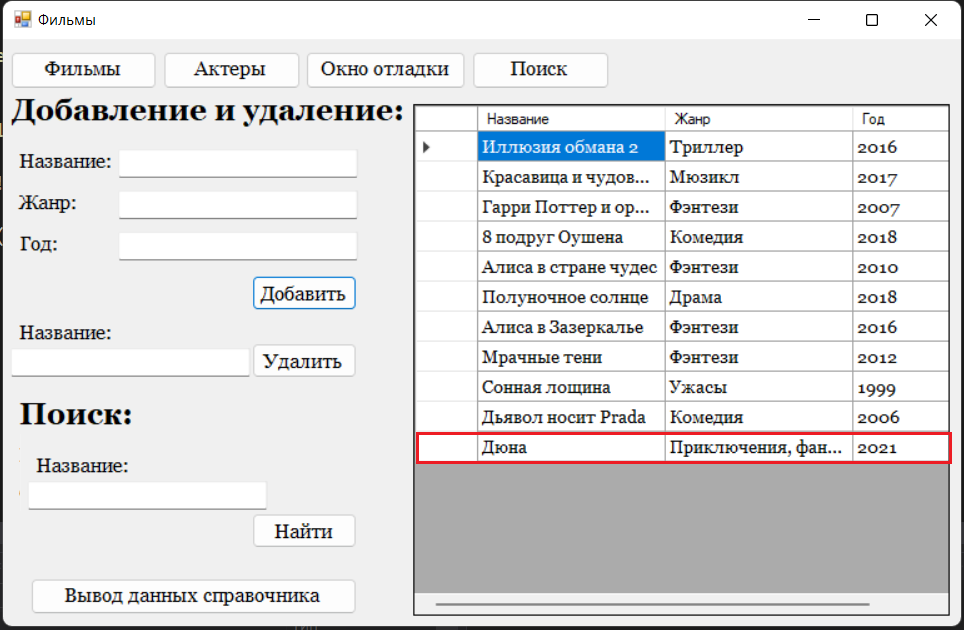


Рисунок 5 - Данные справочника после добавления записи

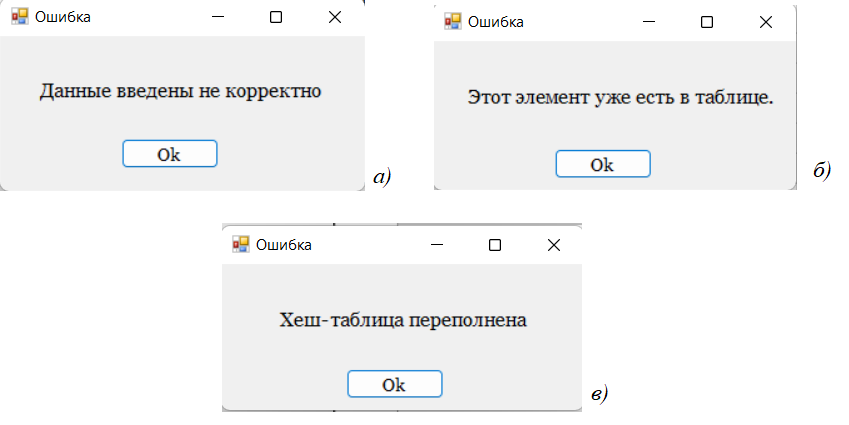
В случае, если введенные данные некорректны (см. п.1.1) или запись с введенными пользователем значениями уже есть в справочнике или хеш-таблица переполнена, то всплывающее окно уведомит пользователя об этом (рисунок 6, а - в).

Рисунок 6 - Сообщения об ошибках: а) если в одном из полей введены не корректные данные; б)добавление уже существующей записи; в)если превышен размер хеш-таблицы.

**Удаление**

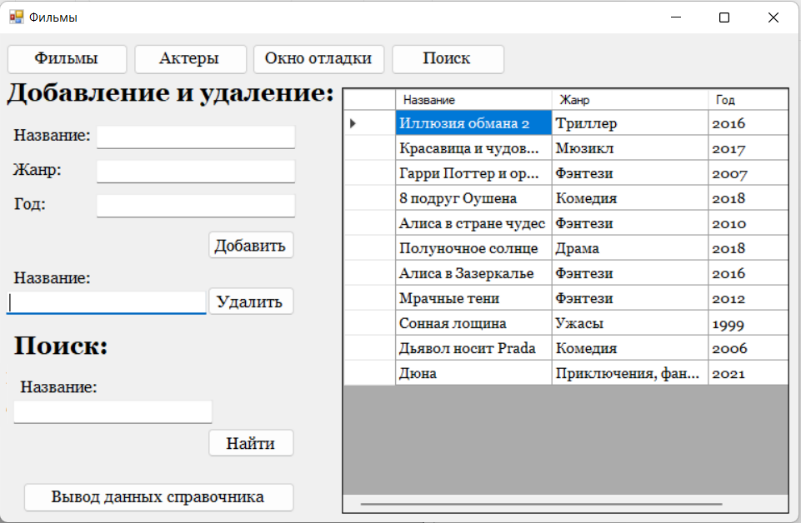
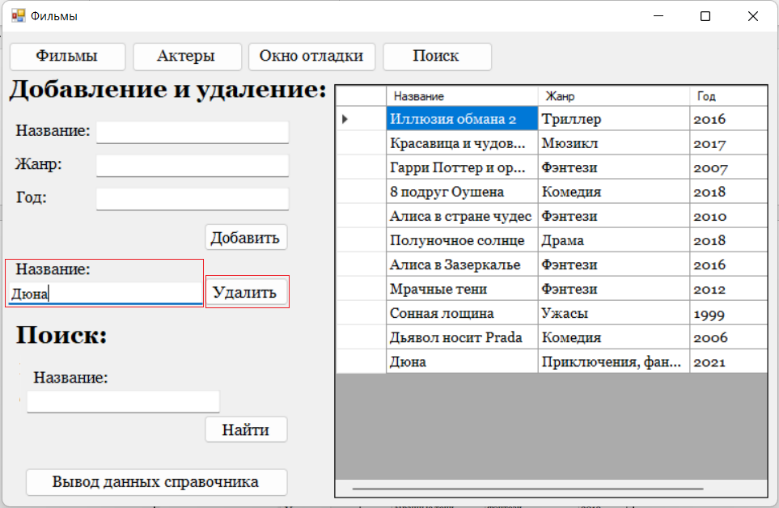
Добавленные записи в справочник «Фильмы» можно удалить. Для демонстрации функции удалим запись, добавленную в предыдущем разделе. Для этого необходимо заполнить поля соотвуетствующими данными и нажать на кнопку «Удалить» (рисунок 7.1 – 7.2 и 8);

Рисунок 7.2 - заполнение данных для удаления

Рисунок 7.1 - окно справочника "Фильмы"

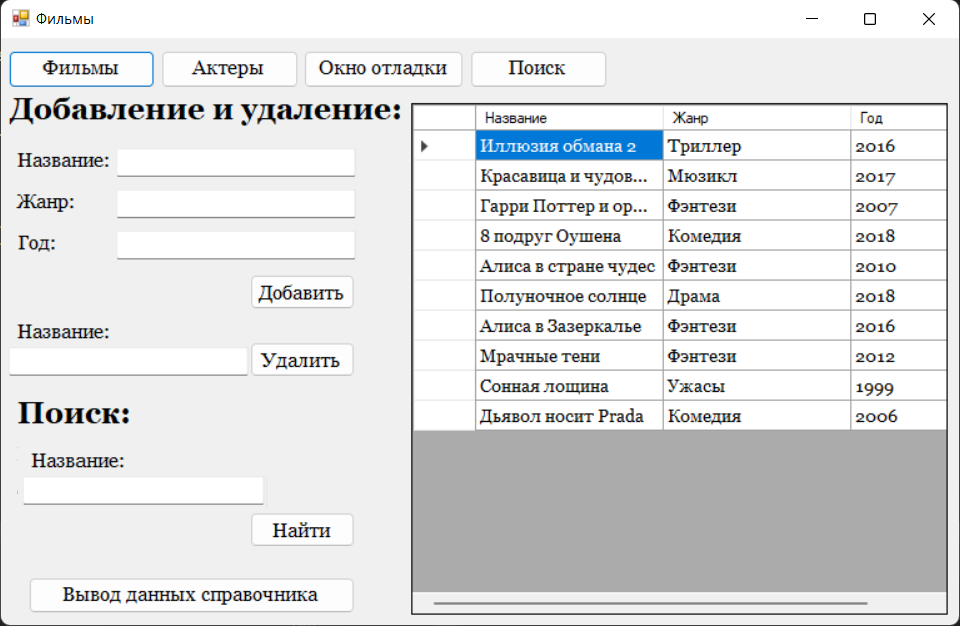


Рисунок 8 - окно справочника "Фильмы" после удаления

В случае, если введенные данные некорректны (см. п.1.1), то всплывающее окно уведомит пользователя об этом (рисунок 9 ).

Рисунок 9 - Ошибка при вводе данных

**Поиск**

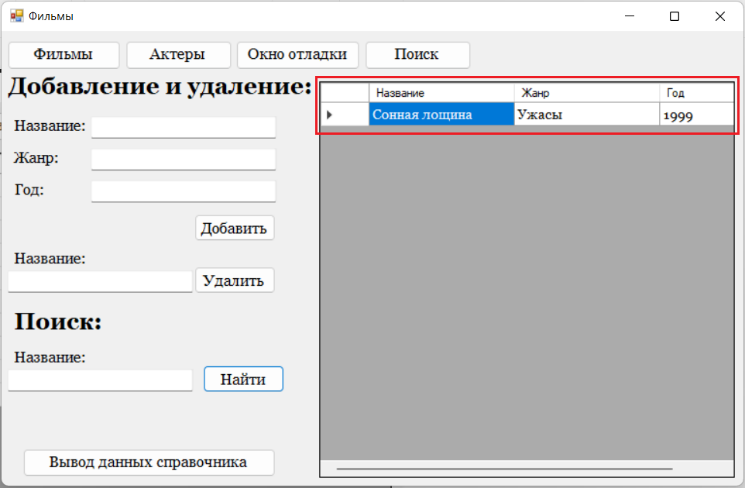
В справочнике «Фильмы», можно выполнить поиск, по дате выхода фильмов. Справочник заполен записями из ПО (см. 1.1). Для этого необходимо заполнить поля соотвуетствующими данными и нажать на кнопку «Найти» (рисунок 10.1 – 10.2);

Рисунок 10.2 - Результат поиска

Рисунок 10.1 - Ввод данных для поиска в справочнике

В случае, если введенные данные некорректны (см. п.1.1), то всплывающее окно уведомит пользователя об этом (рисунок 11).



Рисунок 11 - Вывод ошибки при введении данных

**Вывод в виде Хеш-Таблицы**

В справочнике «Фильмы» хранятся данные, описанные в п.1.1.

Для вывода данных в виде хеш-таблицы необходимо:

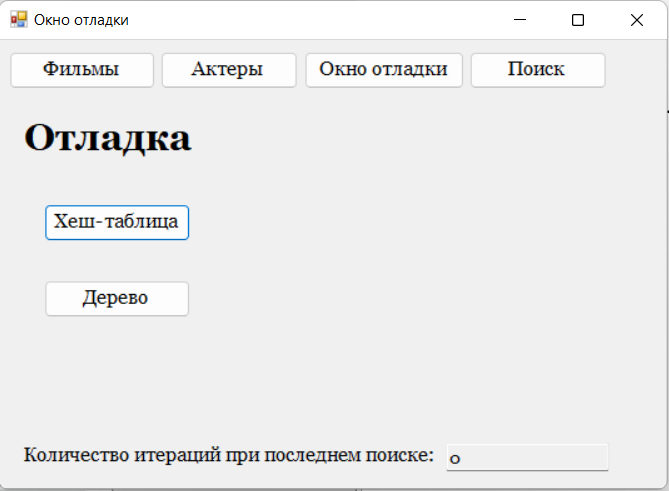
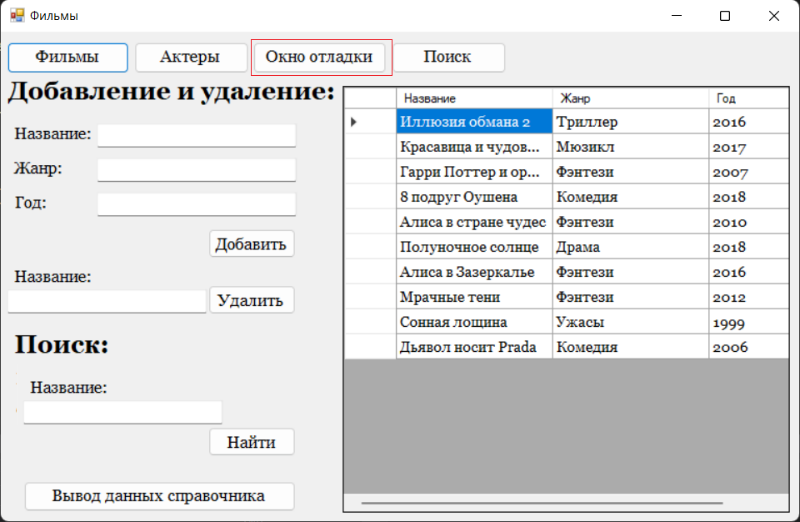
1. Открыть окно отладки (рисунок 12.1 – 12.2);

Рисунок 12.1 - Окно справочника "Фильмы"

Рисунок 12.2 - Окно отладки

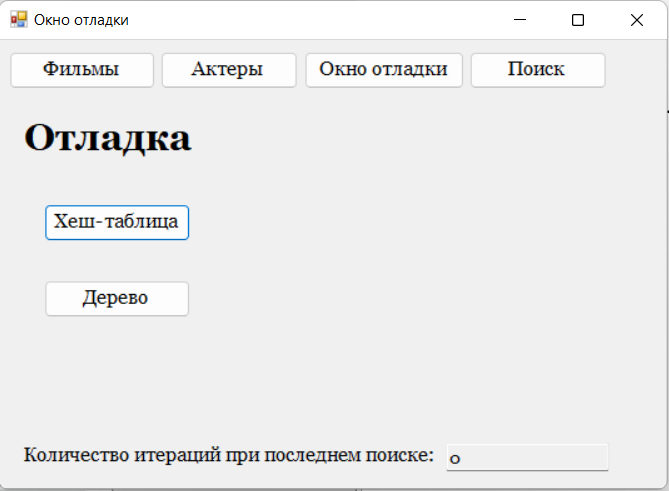
1. Нажать на кнопку «Хеш-таблица» (рисунок 13.1 – 13.3).

Рисунок 13.2 - Окно отладки (кнопка для вывода хеш-таблицы)

Рисунок 13.1 - Окно отладки

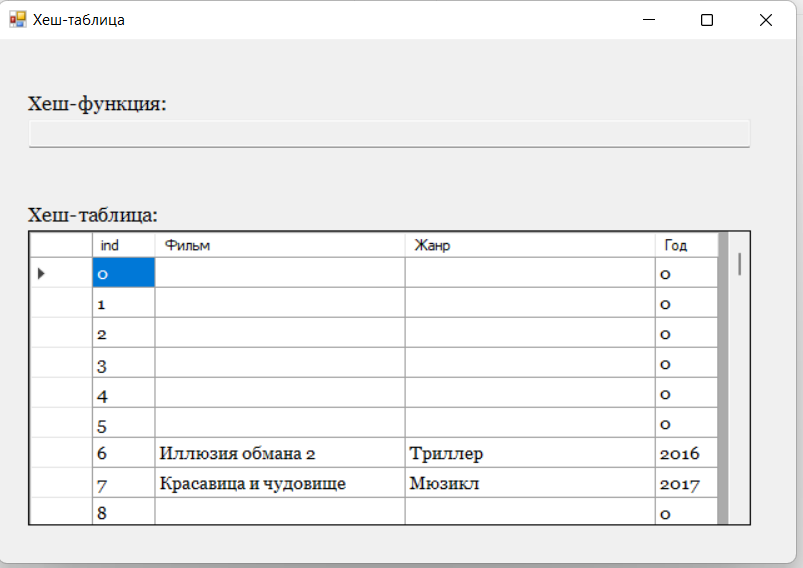


Рисунок 13.3 - Окно Хеш-таблица

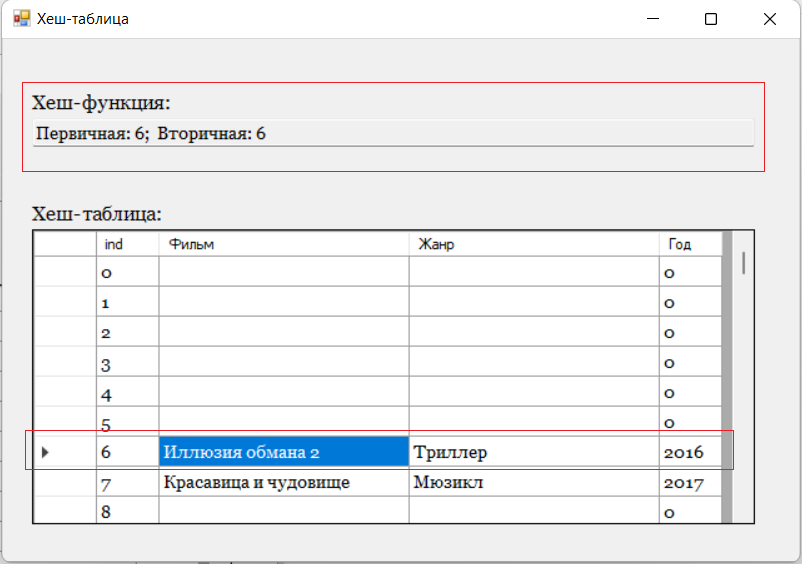
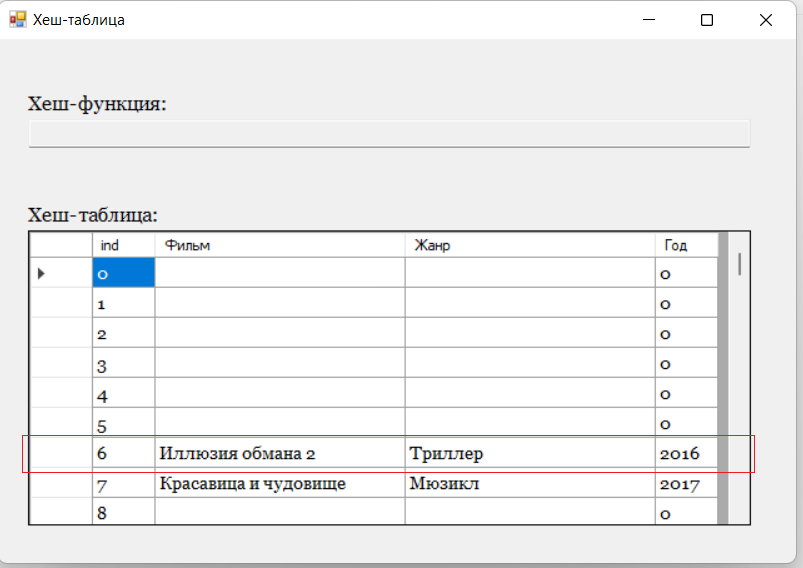
Если нажать на заполненную строку в таблице, то верхней строке, выведется первичная и вторичная хеш-функция, для ключа записанного в этой строке (рисунок 14.1 – 14.2).

Рисунок 14.1 - Окно Хеш-таблица. Выбор строки для вывода первичной и вторичной хеш-функции

Рисунок 14.2 - Окно Хеш-таблица. Вывод первичной и вторичной хеш-функции

### **4.3.2 Работа со справочником «Актеры»**

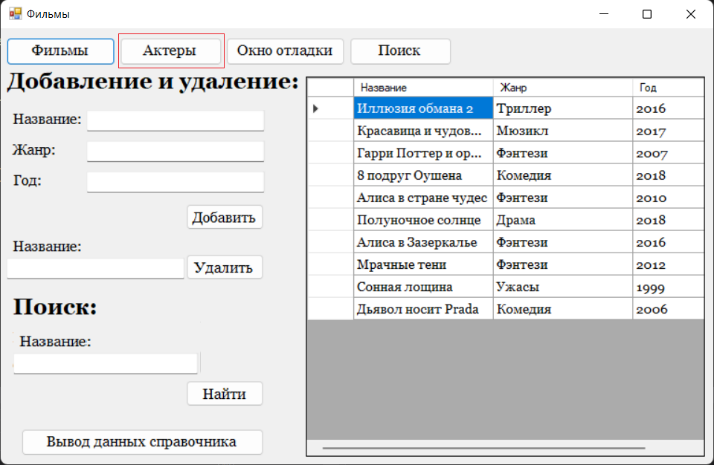
Для работы со справочником «Актеры», нужно перейти в окно «Актеры». Для этого нужно нажать на кнопку «Актеры» (рисунок 15.1 – 15.3).

Рисунок 15.1 - Окно Фильмы. Переход в окно Актеры, для работы с соответствующим справочником

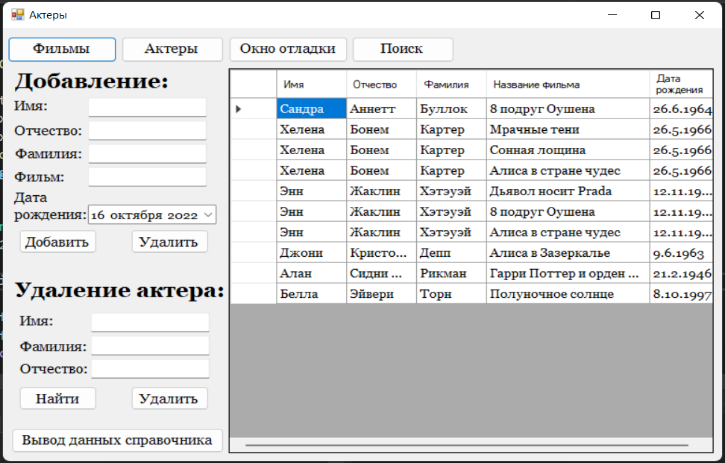


Рисунок 15.2 - Окно Актеры

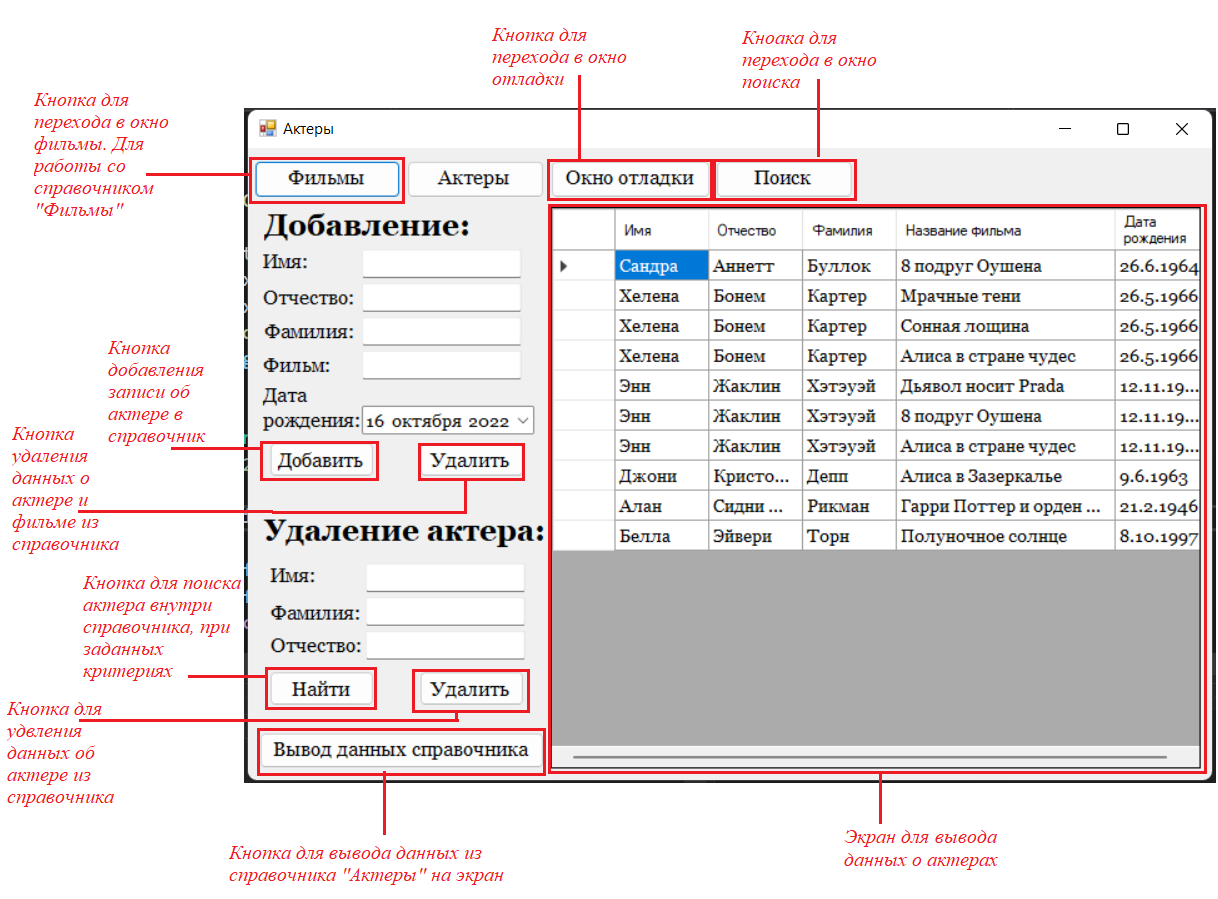
**Добавление**

Рисунок 15.3 - Работа со справочником "Актеры"

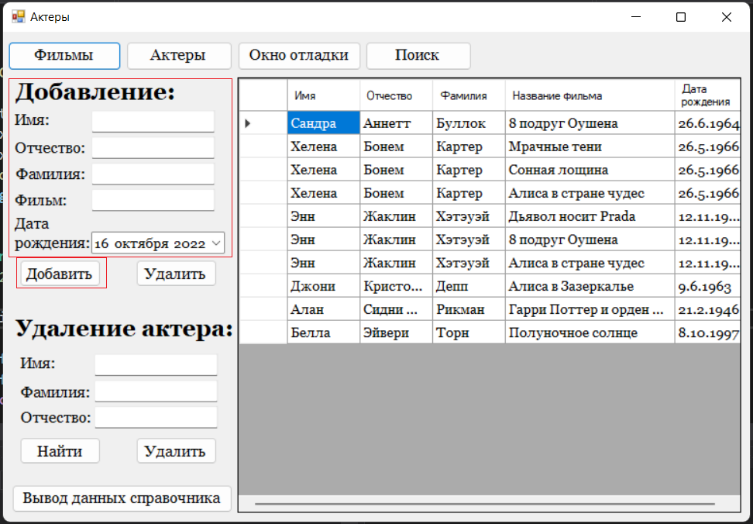
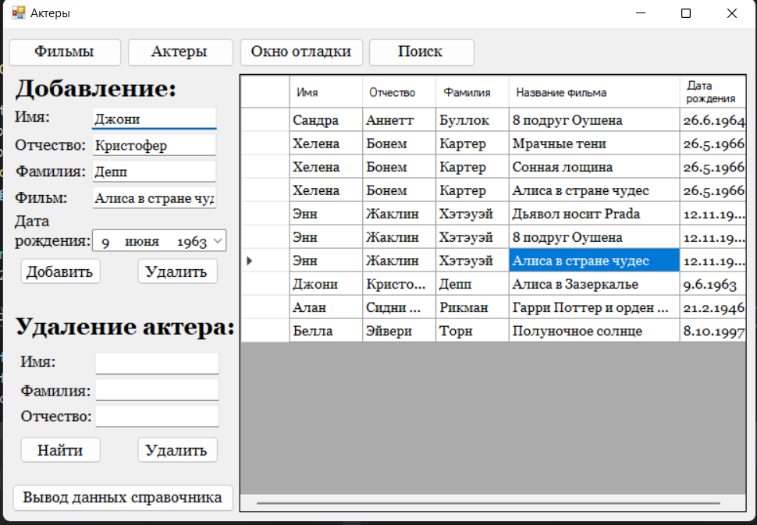
Можно добавлять данные в справочник «Актеры» собственноручно. Для демонстрации будем добавлять данные в справочник, заполненный записями из ПО (см. 1.1). Для этого необходимо заполнить поля соотвуетствующими данными и нажать на кнопку «Добавить» (рисунок 16.1 -16.3);

Рисунок 16.1 - Окно Актеры

Рисунок 16.2 - Окно Актеры. Добавление данных в справочник.

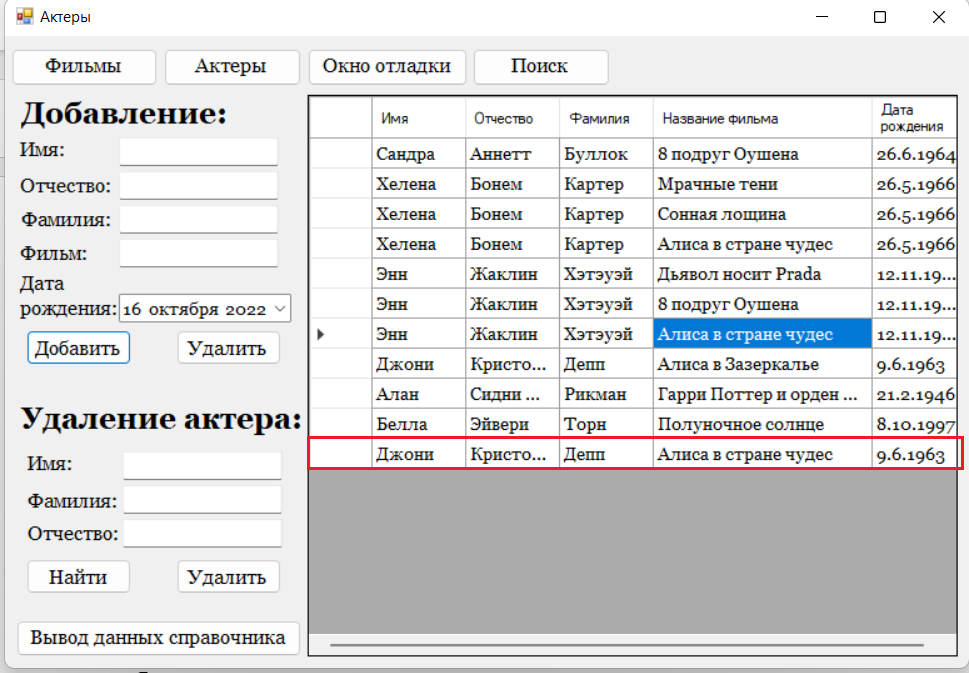


Рисунок 16.3 - Окно Актеры. Добавлен элемент в справочник "Актеры"

В случае, если введенные данные некорректны (см. п.1.1) или фильма не существует в справочнике «Фильмы», то всплывающее окно уведомит пользователя об этом (рисунок 17(а, б)).

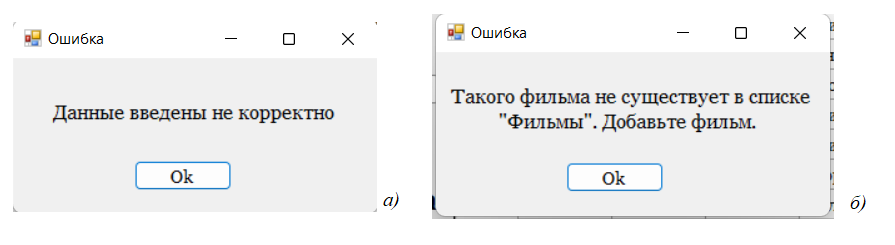


Рисунок 17 - Ошибки при вводе данных: а)данные введены не корректно; б)Нарушена связь между справочниками.

**Удаление**

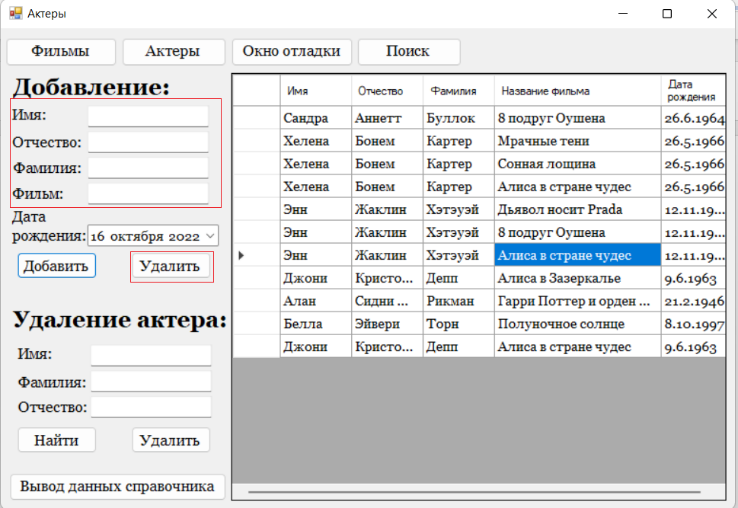
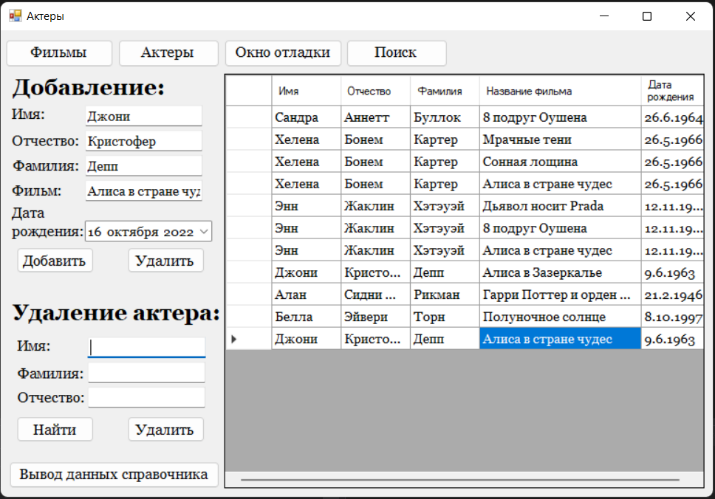
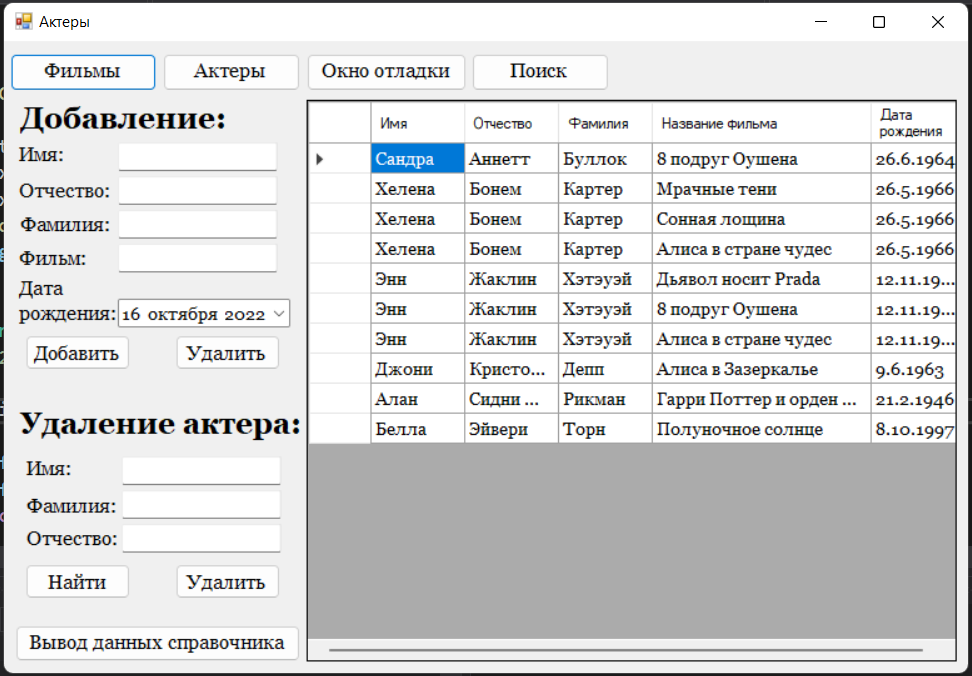
Добавленные записи в справочник «Актеры» можно удалить. Для демонстрации функции удалим запись, добавленную в предыдущем разделе. Для этого необходимо заполнить поля соотвуетствующими данными и нажать на кнопку «Удалить» (рисунок 18.1 – 18.3);

Рисунок 18.3 - Окно Актеры. Результат после удаления

Рисунок 18.2 - Окно Актеры. Удаление по актеру и названию фильма

Рисунок 18.1- Окно Актеры

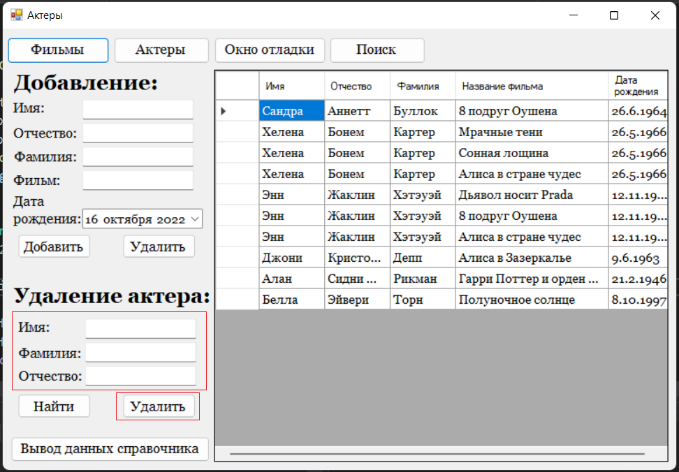
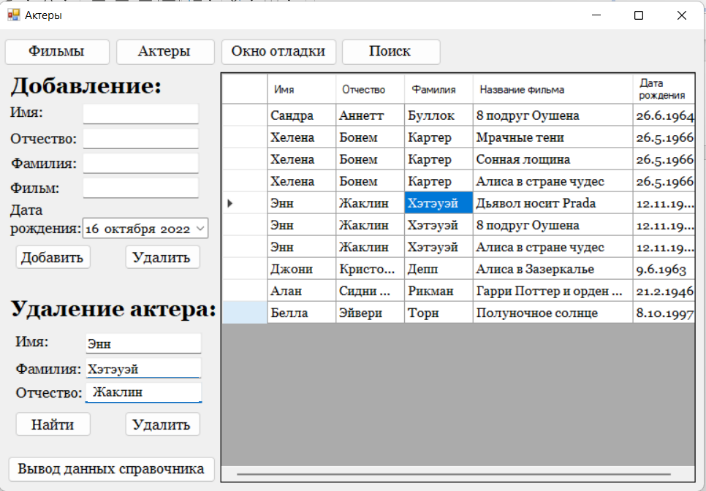
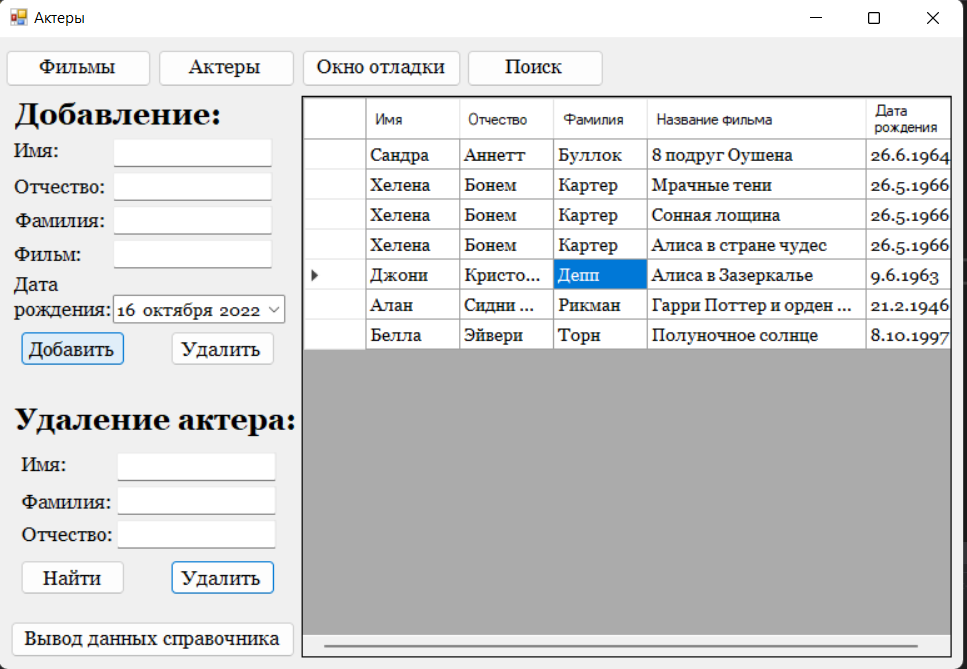
Также существует второй вариант удаления данных в этом справочнике. Можно удалить все данные с одинаковым ключом. Для этого необходимо заполнить поля соотвуетствующими данными и нажать на кнопку «Удалить» (рисунок 19.1 – 19.3);

Рисунок 19.3 - Окно Актеры. Результат удаления

Рисунок 19.2 - Окно Актеры. Удаление по ФИО актера

Рисунок 19.1 - Окно Актеры

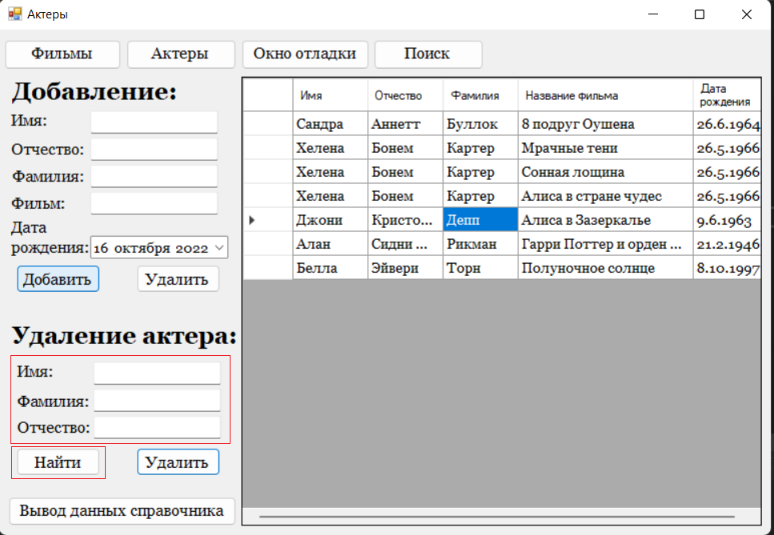
В случае, если введенные данные некорректны (см. п.1.1), то всплывающее окно уведомит пользователя об этом (рисунок 20).

Изображение выглядит как текст

Автоматически созданное описание**Поиск**

Рисунок 20 - Ошибка ввода данных

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описаниеВ справочнике «Актеры», можно выполнить поиск, по ФИО актера. Справочник заполен записями из ПО (см. 1.1). Для этого необходимо заполнить поля соотвуетствующими данными и нажать на кнопку «Найти» (рисунок 21.1 – 21.3);

Рисунок 21.1 - Окно Актеры

Рисунок 21.2 - Окно Актеры. Поиск по ФИО актера

Рисунок 21.3 - Окно Актеры. Результат поиска.

В случае, если введенные данные некорректны (см. п.1.1), то всплывающее окно уведомит пользователя об этом (рисунок 22).

Рисунок 22 - Ошибка ввода данных

**Вывод в виде АВЛ-дерева**

В справочнике «Актеры» хранятся данные, описанные в п.1.1.

Для вывода данных в виде АВЛ-дерева необходимо:

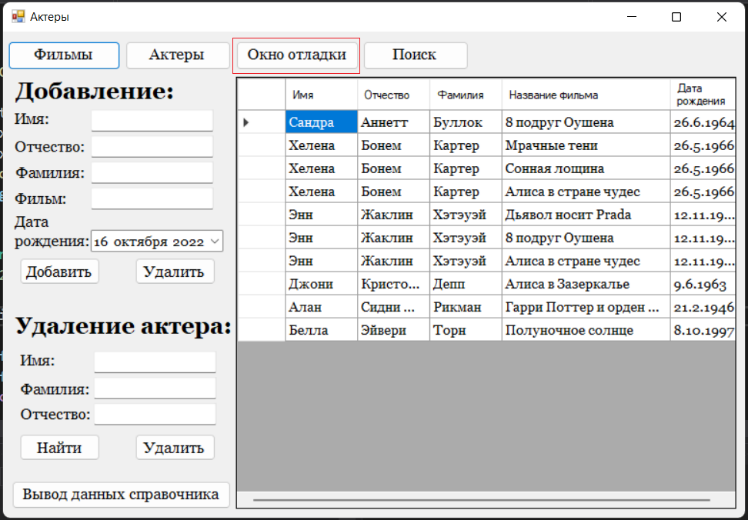
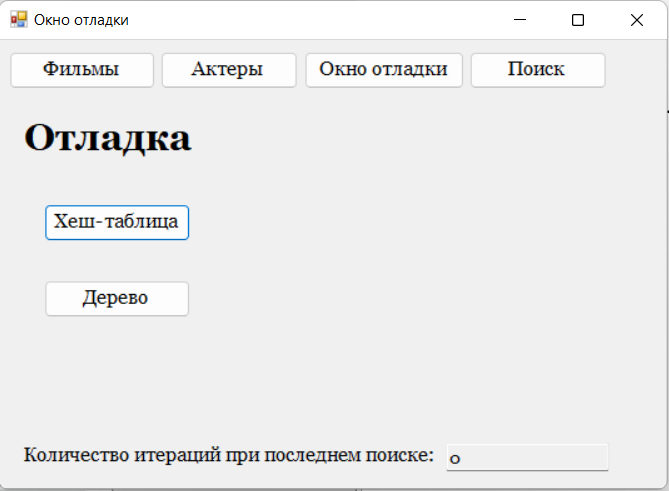
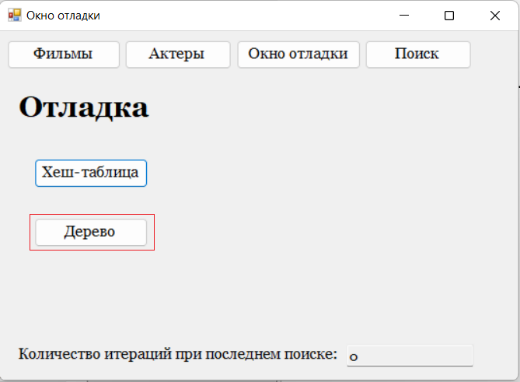
1. Открыть окно отладки (рисунок 23.1 – 23.2);

Рисунок 23.2 - Окно отладки

Рисунок 23.1 - Окно Актеры. Переход в окно отладки

1. Изображение выглядит как текст

   Автоматически созданное описаниеНажать на кнопку «Дерево» (рисунок 24.1 – 24.3).

Рисунок 24.2 - Окно отладки. Просмотр дерева.

Рисунок 24.1 - Окно отладки

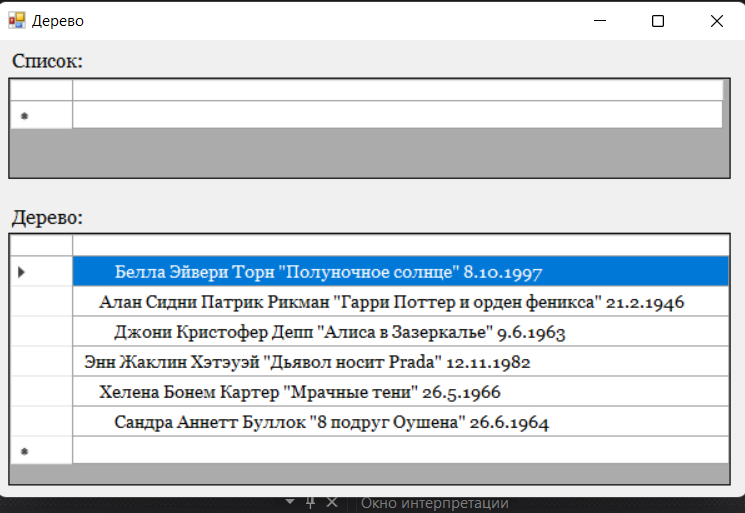


Рисунок 24.3 - Окно АВЛ-дерева

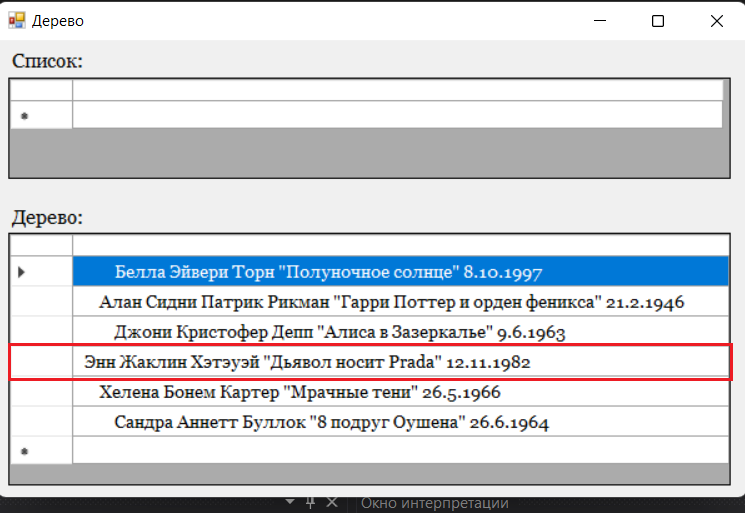
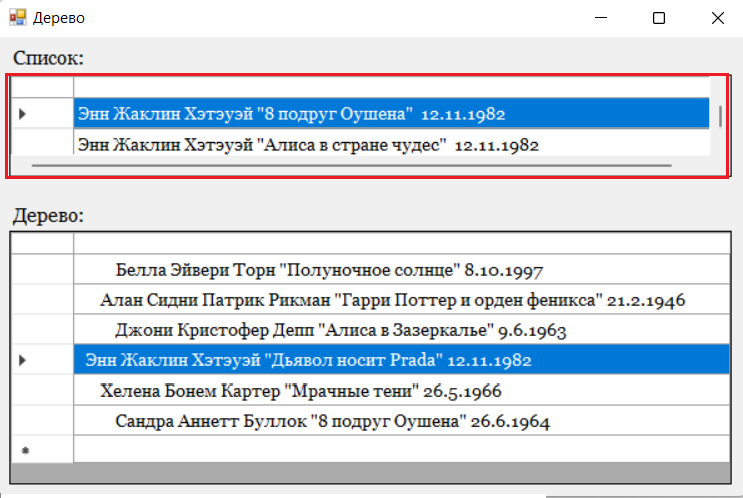
Если нажать на заполненную строку в нижней таблице, то верхней таблице, выведется список этого узла дерева, если он существует (рисунок 25.1 – 25.2).

Рисунок 25.2- Список узла дерева

Рисунок 25.1 -Выбор строки в дереве для просмотра списка

### **4.3.3 Формирование отчета**

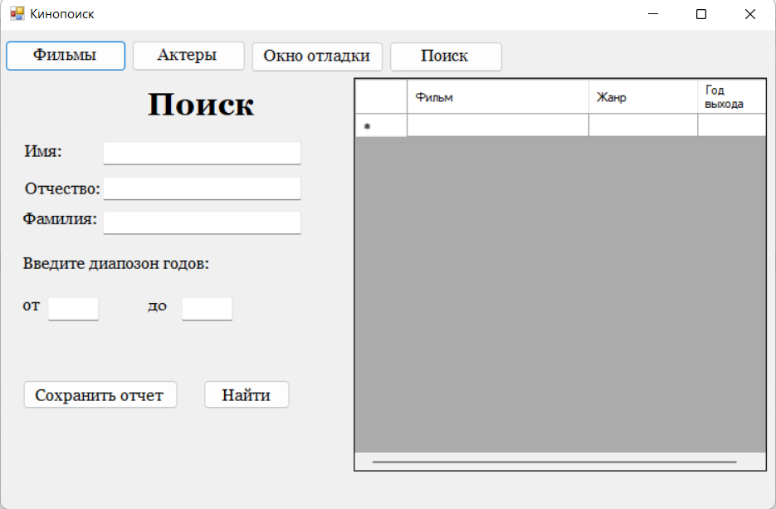
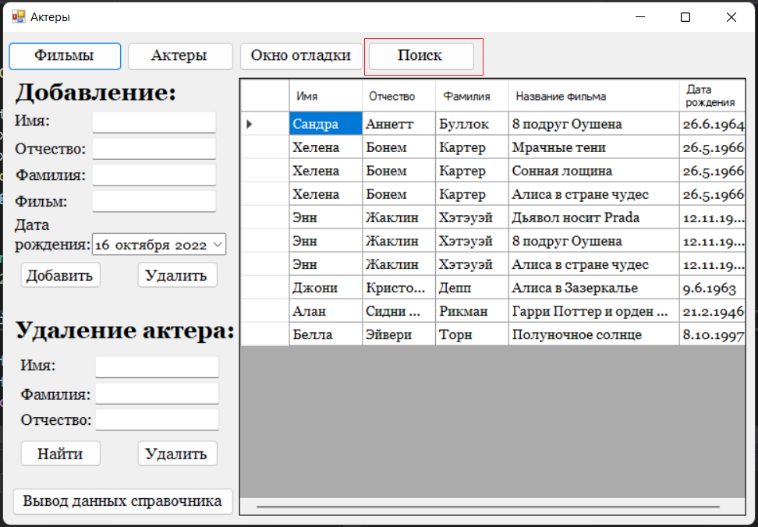
Для формирования отчета нужно перейти в окно «Поиск», для этого нужно нажать на кнопку «Поиск» (рисунок 26.1 – 26.2).

Рисунок 26.2 - Окно поиска

Рисунок 26.1 - Переход из окна Актеры в окно поиска

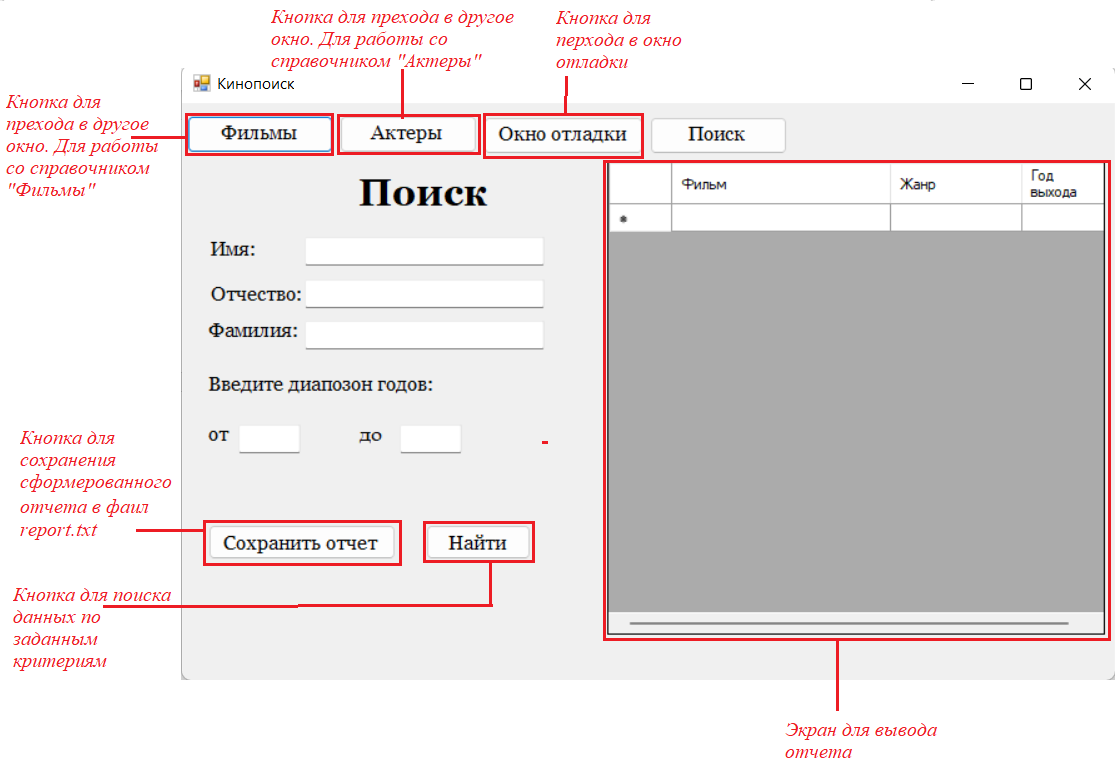
Открывается окно, в котором можно вывести фильмы, в которых участвовал актер, год выхода должен попадать в диапазон, заданный пользователем, а также сохранить полученный отчет в файл (рисунок 27).

Рисунок 27 - Описание она поиска

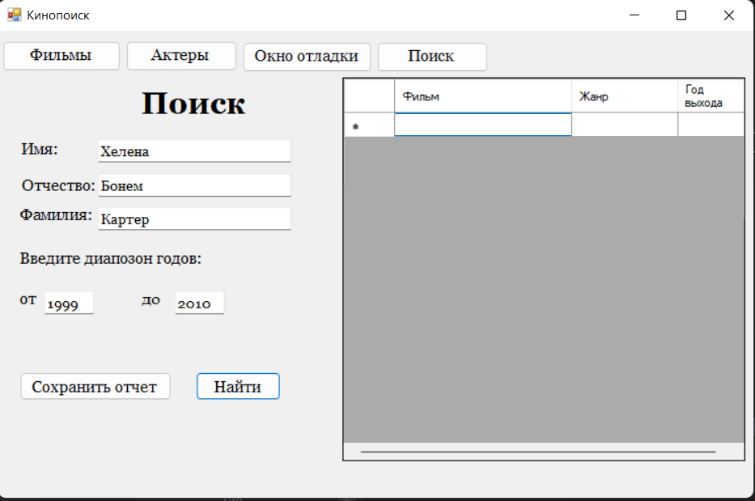
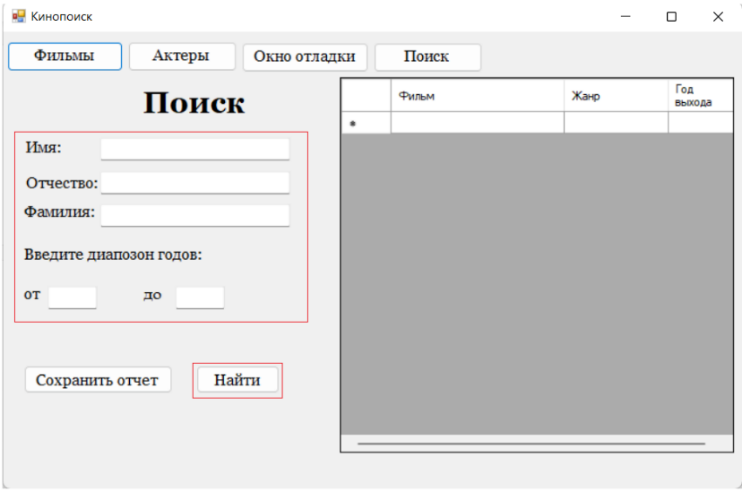
Для этого нужно в открывшемся окне ввести соответствующие данные и нажать на кнопку «Найти». Если в справочнике «Актеры» существуют актер, то выведутся фильмы, год выхода которых входит в диапазон, заданный пользователем (рисунок 28.1 – 28.3).

Рисунок 28.2 - Окно поиска с заполненными данными для формирования отчета

Рисунок 28.1 - Окно поиска

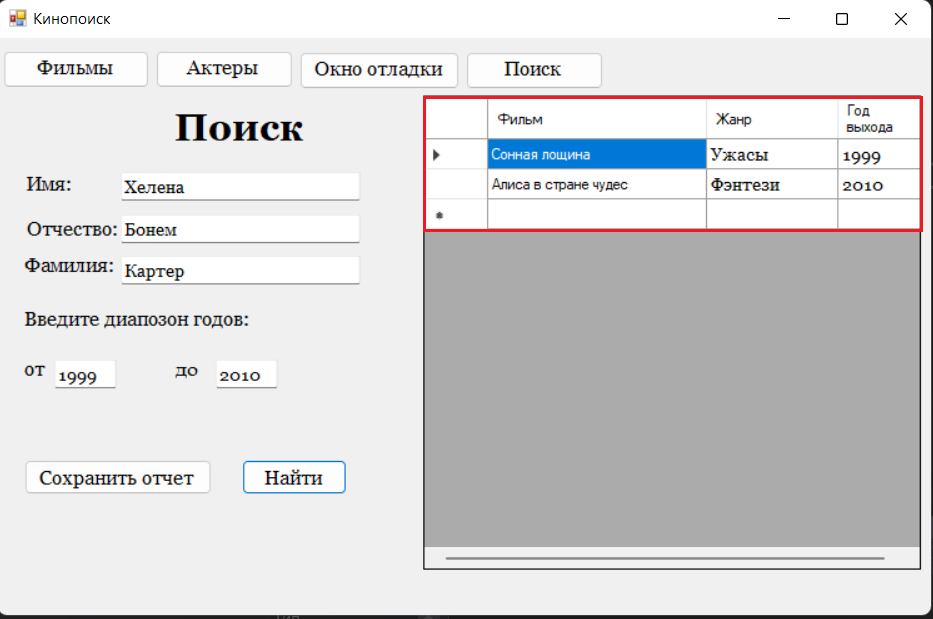


Рисунок 28.3 - Результаты поиска

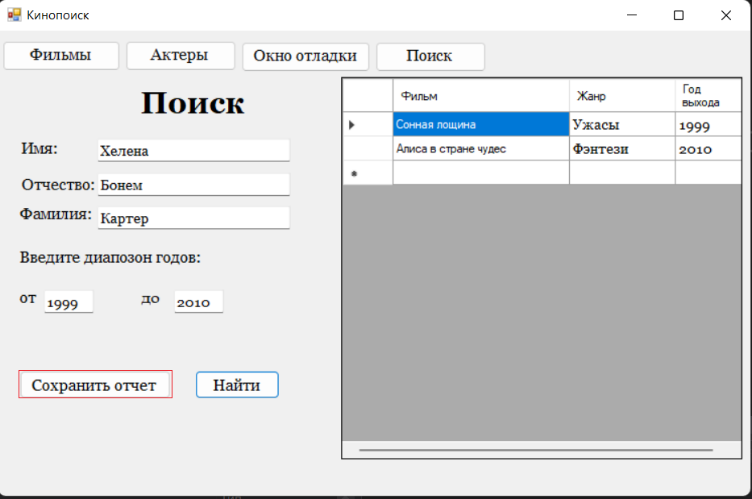
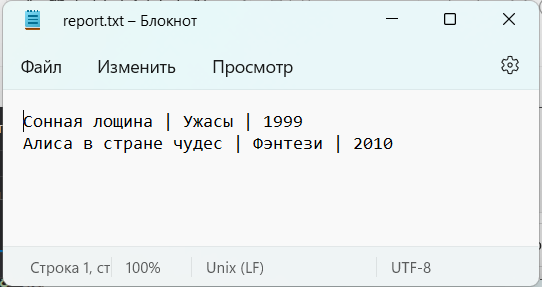
Для записи сформированного отчета файл, нужно нажать на кнопку «Сохранить отчет» (рисунок 29).

Рисунок 29 - Сохранение отчета в файл

В случае, если введенные данные некорректны (см. п.1.1), то всплывающее окно уведомит пользователя об этом (рисунок 30).



Рисунок 30 - Ошибка ввода данных

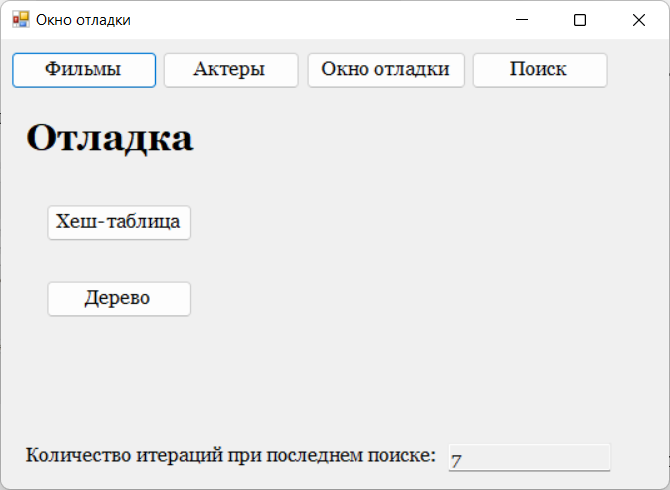
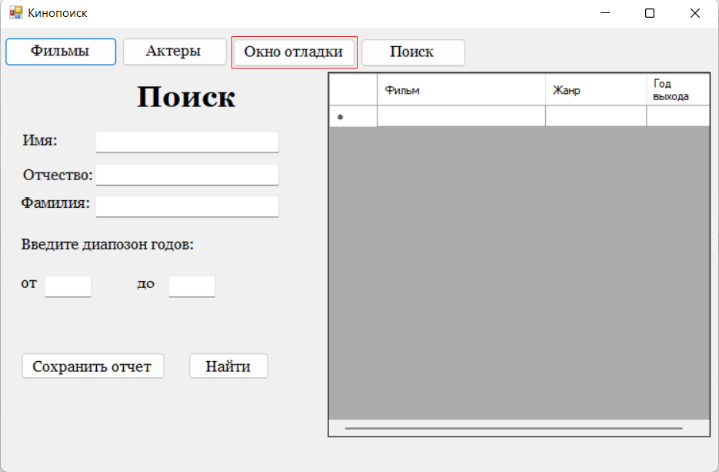
Для просмотра количества итераций при поиске нужно перейти в окно отладки. Для этого нажать на кнопку «Окно отладки» (рисунок 31).

Рисунок 31 - Переход из окна поиска в окно отладки

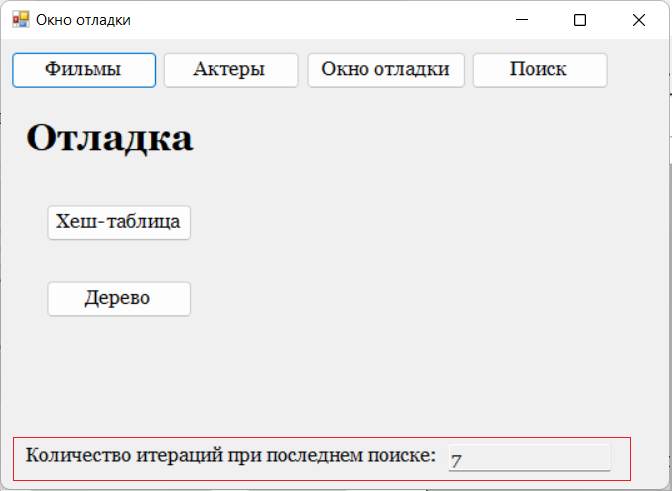
Внизу отобразится строка, в которой записано количество итераций при последнем поиске (рисунок 32).

Рисунок 32 - Количество итераций при поиске

4.4 Тестирование

Таблица 5 – Тестирование работы со справочником «Фильмы»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | |
| Хеш-таблица (метод открытой адресации) | Запись о фильме (Название, жанр, год) | Хеш-таблица (метод открытой адресации) | Сообщение |
| Добавление | | | | | |
| 1 | Добавление некорректных данных | - | 8 подруг Оушена | комедия | 2018 | - | «Данные введены некорректно» |
| - | 8 подруг Оушена | sdfdgfdg | 2018 | - | «Данные введены некорректно» |
| - | 8 подруг Оушена | Комедия | 1900 | - | «Данные введены некорректно» |
| - | 8 подруг Оушена | Комедия | 2023 | - | «Данные введены некорректно» |
| - | | Комедия | 2018 | - | «Данные введены некорректно» |
| - | 8 подруг Оушена | | 2018 | - | «Данные введены некорректно» |
| - | 8 подруг Оушена | Комедия | | - | «Данные введены некорректно» |
| 2 | Добавление корректных данных | - | 8 подруг Оушена | Комедия | 2018 | 8 подруг Оушена | Комедия | 2018 | - |
| 3 | Проверка уникальности ключа | 8 подруг Оушена | Комедия | 2018 | 8 подруг Оушена | Комедия | 2018 | 8 подруг Оушена | Комедия | 2018 | «Этот элемент уже есть в таблице» |
| 8 подруг Оушена | Комедия | 2018 | 8 подруг Оушена | Криминал | 2018 | 8 подруг Оушена | Комедия | 2018 | «Этот элемент уже есть в таблице» |
| 8 подруг Оушена | Комедия | 2018 | 8 подруг Оушена | Комедия | 2019 | 8 подруг Оушена | Комедия | 2018 | «Этот элемент уже есть в таблице» |
| 4 | Добавление при коллизии | Мрачные тени | Фэнтези | 2012 | Алиса в Зазеркалье | Фэнтези | 2016 | Мрачные тени | Фэнтези | 2012  Алиса в Зазеркалье | Фэнтези | 2016  (Хеш совпал, запись с ключом «Oen» была добавлена через одну ячейку от записи с ключом «One») | - |
| Поиск | | | | | |
| 5 | Запись не существует | - | Алиса в Зазеркалье | - | - |
| 6 | Запись существует | Алиса в Зазеркалье | Фэнтези | 2016 | Алиса в Зазеркалье | Алиса в Зазеркалье | Фэнтези | 2016 | - |
| 7 | Поиск при коллизии | Мрачные тени | Фэнтези | 2012  Алиса в Зазеркалье | Фэнтези | 2016 | Алиса в Зазеркалье | Мрачные тени | Фэнтези | 2012  Алиса в Зазеркалье | Фэнтези | 2016 | - |
| Удаление | | | | | |
| 8 | Запись не существует | - | Алиса в Зазеркалье | - | - |
| 9 | Запись существует | Алиса в Зазеркалье | Фэнтези | 2016 | Алиса в Зазеркалье | - | - |
| 10 | Удаление при коллизии | Мрачные тени | Фэнтези | 2012  Алиса в Зазеркалье | Фэнтези | 2016 | Алиса в Зазеркалье | Мрачные тени | Фэнтези | 2012 | - |

Таблица 6 – Тестирование работы со справочником «Актеры»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные | |
| АВЛ-дерево | Запись об актере  (ФИО, название фильма, дата рождения) | АВЛ-дерево | Сообщение |
| Добавление | | | | | |
| 1 | Добавление некорректных данных | - | энн | Жаклин | Хэтэуэй | 8 подруг Оушена | 12.11.1982 | - | «Данные введены некорректно» |
| - | Энн | жаклин | Хэтэуэй | 8 подруг Оушена | 12.11.1982 | - | «Данные введены некорректно» |
| - | Энн | Жаклин | хэтэуэй | 8 подруг Оушена | 12.11.1982 | - | «Данные введены некорректно» |
| - | ghjgh | Жаклиhн | Хэтhjэуэй | 8 подруг Оушена | 12.11.1982 | - | «Данные введены некорректно» |
| - | - | - | «Данные введены некорректно» |
|  | Энн | | Хэтэуэй | 8 подруг Оушена | 12.11.1982 |  | «Данные введены некорректно» |
|  | Энн | Жаклин | | 8 подруг Оушена | 12.11.1982 |  | «Данные введены некорректно» |
|  | Энн | Жаклин | Хэтэуэй | | 12.11.1982 |  | «Данные введены некорректно» |
| - | | Жаклин | Хэтэуэй | 8 подруг Оушена | 12.11.1982 | - | «Данные введены некорректно» |
| 2 | Добавление корректных данных | - | Энн | Жаклин | Хэтэуэй | 8 подруг Оушена | 12.11.1982 | Энн | Жаклин | Хэтэуэй | 8 подруг Оушена | 12.11.1982 | - |
| 4 | Добавление при коллизии | Энн | Жаклин | Хэтэуэй | 8 подруг Оушена | 12.11.1982 | Энн | Жаклин | Хэтэуэй | Алиса в стране чудес| 12.11.1982 | Энн | Жаклин | Хэтэуэй | 8 подруг Оушена | 12.11.1982  Энн | Жаклин | Хэтэуэй | Алиса в стране чудес| 12.11.1982  (значение поля «ФИО» обоих ключей совпадает, а остальные поля различаются, поэтому вторая запись будет добавлена в конец динамического двусвязного кольцевого списка) | - |
| Поиск | | | | | |
| 5 | Запись не существует | - | Энн | Жаклин | Хэтэуэй | - | - |
| 6 | Запись существует | Энн | Жаклин | Хэтэуэй | Алиса в стране чудес| 12.11.1982 | Энн | Жаклин | Хэтэуэй | Энн | Жаклин | Хэтэуэй | Алиса в стране чудес| 12.11.1982 | - |
| 7 | Поиск при коллизии | Энн | Жаклин | Хэтэуэй | 8 подруг Оушена | 12.11.1982  Энн | Жаклин | Хэтэуэй | Алиса в стране чудес| 12.11.1982 | Энн | Жаклин | Хэтэуэй | Энн | Жаклин | Хэтэуэй | 8 подруг Оушена | 12.11.1982  Энн | Жаклин | Хэтэуэй | Алиса в стране чудес| 12.11.1982 | - |
| Удаление | | | | | |
| 8 | Запись не существует | - | Энн | Жаклин | Хэтэуэй | 8 подруг Оушена| 12.11.1982 | - | - |
| - | Энн | Жаклин | Хэтэуэй | - | - |
| 9 | Запись существует | Энн | Жаклин | Хэтэуэй | 8 подруг Оушена | 12.11.1982 | Энн | Жаклин | Хэтэуэй | 8 подруг Оушена| 12.11.1982 | - | - |
| Энн | Жаклин | Хэтэуэй | 8 подруг Оушена | 12.11.1982  Энн | Жаклин | Хэтэуэй | Алиса в стране чудес| 12.11.1982 | Энн | Жаклин | Хэтэуэй | - | - |
| 10 | Удаление при коллизии | Энн | Жаклин | Хэтэуэй | 8 подруг Оушена | 12.11.1982  Энн | Жаклин | Хэтэуэй | Алиса в стране чудес| 12.11.1982 | Энн | Жаклин | Хэтэуэй | 8 подруг Оушена| 12.11.1982 | Энн | Жаклин | Хэтэуэй | Алиса в стране чудес| 12.11.1982 | - |

Таблица 7 – Тестирование работы структуры данных АВЛ-дерево

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные |
| АВЛ-дерево | Запись об актере  (ФИО, название фильма, дата рождения) | АВЛ-дерево |
| Добавление | | | | |
| 1 | Добавление в пустое дерево | - | Энн | Жаклин | Хэтэуэй | 8 подруг Оушена| 12.11.1982 |  |
| 2 | Добавление в непустое дерево | Энн | Жаклин | Хэтэуэй | 8 подруг Оушена| 12.11.1982 | Хелена | Бонем | Картер | Сонная лощина | 26.05.1966 |  |
| 3 | Добавление повторяющегося элемента |  | Хелена | Бонем | Картер | Мрачные тени| 26.05.1966 |  |
| 4 | Добавление с последующим левым поворотом |  | Белла | Эйвери | Торн | Полуночное солнце| 08.10.1997 | Изображение выглядит как текст  Автоматически созданное описание |
| 5 | Добавление с последующим правым поворотом | Изображение выглядит как текст  Автоматически созданное описание | Сандра | Аннетт | Буллок | 8 друзей Оушена| 26.06.1964 |  |
| 6 | Добавление с последующим лево-правым поворотом |  | Миа | Пр | Васиковска | Алиса в стране чудес| 26.06.1991 |  |
| 7 | Добавление с последующим право-левым поворотом |  | Рисс | Пр | Иванс | Алиса в стране чудес| 25.02.1984 |  |
| Удаление | | | | |
| 8 | Удаление из пустого дерева | - | Сандра | Аннетт | Буллок | 8 друзей Оушена| 26.06.1964 | - |
| 9 | Удаление из непустого дерева эл-та которого нет |  | Сандра | Аннетт | Буллок | 8 друзей Оушена| 26.06.1964 |  |
| 10 | Удаление из непустого дерева эл-та который есть |  | Энн | Жаклин | Хэтэуэй | 8 подруг Оушена| 12.11.1982 |  |
| 11 | Удаление корня дерева |  | Алан | Сидни Патрик | Рикман | Гарри Поттер и орден феникса| 21.02.1946 |  |

Таблица 8 – Тестирование структуры данных хеш-таблица

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Описание тестовой ситуации | | Входные данные | | Выходные данные |
| Хеш-таблица (метод открытой адресации) | Запись о фильме (Название, жанр, год) | Хеш-таблица (метод открытой адресации) |
| Добавление | | | | |
| 1 | Добавление в пустую хеш-таблицу | - | Мрачные тени | Фэнтези|2012 |  |
| 2 | Добавление в непустую хеш-таблицу |  | Сонная лощина| Ужасы| 1999 |  |
| 3 | Добавление повторяющегося элемента |  | Мрачные тени | Фэнтези|2012 |  |
| 4 | Добавление с таким же значением первичного хеша |  | Алиса в Зазеркалье | Фэнтези| 2016 |  |
| Удаление | | | | |
| 6 | Удаление из пустой хеш-таблицы | - | Алиса в Зазеркалье | - |
| 7 | Удаление из непустой хеш-таблицы, данная запись есть в таблице |  | Сонная лощина |  |
| 8 | Удаление из непустой хеш-таблицы, данная запись отсутствует в таблице |  | Алиса в Зазеркалье |  |
| 9 | Удаление с синонимичным ключом |  | Алиса в Зазеркалье |  |

# Заключение

Целью курсового проекта было: разработать информационную систему «Кинопоиск».

Цель достигнута. Для достижения поставленной цели были выполнены следующие задачи:

1. Проведён анализ предметной области проекта «Кинопоиск»;

2. Изучены теоретические основы методов построения справочников;

3. Определены требования к информационной системе;

4. Информационная система была реализована и протестирована;

* + - * 1. Дополнительно для реализации системы был изучен язык C#, а также изучен .NET Framework 4.8.

# Список литературы

1. Кормен, Т. Х. Алгоритмы: построение и анализ / Лейзерсон Ч. И., Ривест Р. Л., Штайн К. – 3-е изд.: Пер. с англ. – М.: ООО «И.Д.Вильямс», 2013. – 1328 c.: ил. парал. тит. англ.
2. Седжвик Р. Фундаментальные алгоритмы на С++. Анализ/Структуры данных/Сортировка/Поиск. / Роберт Седжвик. – Киев : Издательство «ДиаСофт», 2001. – 688 с
3. Никлаус Вирт, Алгоритмы и структуры данных. Новая версия для Оберона + CD / Пер. с англ. Ткачев Ф. В. – М.: ДМК Пресс, 2010. – 272 с.:ил.