СОДЕРЖАНИЕ

ВЕДЕНИЕ

В данной работе производится моделирование информационной системы «Кинотеатры».

Актуальность работы состоит в том, что она рассматривает модель информирования сотрудников справочной службы кинотеатров города. Имея доступ к базам сведений кинотеатров, они без проблем могут задать нужный запрос и получить на них ответ, что в свою обеспечивает больший для клиентов кинотеатров, а значит и повышает эффективность работы самих кинотеатров.

Объектом разработки является программа, написанная на языке C++, эмулирующая работу справочной службы кинотеатров.

Предмет разработки – объектная модель информационной системы.

Целью курсового проектирования является разработка программного продукта – системы имитационного моделирования.

Задачи курсового проектирования:

– провести объектно-ориентированный анализ системы;

– построить объектную модель системы;

– спроектировать алгоритм работы системы;

– реализовать программу на языке C++, моделирующую данную систему.

Результат курсового проектирования: работающая программа, осуществляющая человеко-машинное взаимодействие согласно техническому заданию.

1 ПОСТАНОВКА ЗАДАЧИ

1.1 Описание предметной области

При помощи справочной службы кинотеатров города значительно повышается комфортабельность процесса поиска нужного фильма, сеанса или кинотеатра. Вместо неопределенности и траты времени, необходимо лишь сформировать нужный клиенту запрос в виде определенного набора данных и обратиться к работникам информационной службы. Получив необходимые данные, они за кратчайшие сроки с совершенной точностью найдут и предложат клиенту все подходящие варианты.

Изначально данная система имеет доступ ко всем данным кинотеатров города и фильмам в целом. Благодаря чему, возможна не только демонстрация необходимых сведений, но и работа с ними. Доступны такие возможности, как добавить, удалить или же редактировать ту или иную информацию, что обеспечивает динамичность данной системы.

Благодаря наличию легкодоступного и интуитивно понятного интерфейса, человек работающий с данной системой не будет иметь каких-либо трудностей или неопределенностей с взаимодействием с программой. Запустив систему, на экране откроется консоль с дальнейшими инструкциями. Изучив их пользователь может вводить команды согласно инструкциям. Если пользователь программы случайно ошибется, то программа предусмотрит и укажет об этом, исключая неверный ход событий приложения.

Главные объекты данной системы — кинотеатры, фильмы и репертуары. Все они образуют целостную систему благодаря взаимосвязям друг с другом. Фильмы хранятся в репертуарах, а репертуары — кинотеатрах. Работникам дана возможность демонстрации как общих данных, так и отдельных благодаря поиску по нужным свойствам объектов.

При эксплуатации системы, пользователем не стоит заботиться о сохранности данных или их утечке. Программа автоматически загружает все данные при ее запуске и сохраняет при выходе из нее. Все эти и другие особенности данной системы обеспечивает наибольшую эффективность работы справочной службы

1.2 Цели и задачи курсового проектирования

Цель курсового проектирования – проанализировать, спроектировать и реализовать модель информациоонной системы «Кинотеатры ».

При разработке системы следует выделить несколько задач:

– произвести объектно-ориентированнный анализ предметной области;

– спроектировать систему, осуществляющую взаимодействие между пользователем и её составными частями, построить соответсвующие UML-диаграммы;

– создать функциональный программный продукт на основе произведённого анализа и проектирования, соответствующий техническому заданию;

– оформить руководство пользователя согласно ГОСТам.

2 ВЫБОР МЕТОДА РЕШЕНИЯ

2.1 Описание ОО подхода

Определение классов и объектов – одна из самых сложных задач объектно-ориентированного проектирования.

Класс – это множество объектов, связанных общностью структуры и поведения. Любой объект является экземпляром класса.

Основополагающей идеей объектно-ориентированного подхода является объединение данных и действий, производимых над этими данными, в единое целое, которое называется объектом.

Объект имеет определённые свойства. Состояние объекта задаётся значениями его признаков. Объект «знает», как решать определённые задачи, то есть располагает методами решения. Программа, написанная с использованием ООП, состоит из объектов, которые могут взаимодействовать между собой. Ранее отмечалось, что программная реализация объекта представляет собой объединение данных и процедур их обработки. Переменные объектного типа называют экземплярами объекта. В отличие от типа «запись», объектный тип содержит не только поля, описывающие данные, но также процедуры и функции, описания которых содержится в описании объекта. Эти процедуры и функции называют методами. Методам объекта доступны его поля. Следует отметить, что методы и их параметры определяются в описании объекта, а их реализация даётся вне этого описания, в том месте программы, которое предшествует вызову данного метода. В описании объекта фактически содержаться лишь шаблоны обращения к методам, которые необходимы компилятору для проверки соответствия количества параметров и их типов при обращении к методам.

Если необходимо считать какие-либо данные объекта, нужно вызвать соответствующий метод, который выполнит считывание и возвратит требуемое значение.

Прямой доступ к данным невозможен. Данные сокрыты от внешнего воздействия, что защищает их от случайного изменения. Говорят, что данные и методы инкапсулированы.

Инкапсуляция является важнейшим свойством объектов, на котором строится объектно-ориентированное программирование. Инкапсуляция заключается в том, что объект скрывает в себе детали, которые несущественны для использования объекта. В традиционном подходе к программированию с использованием глобальных переменных программист не был застрахован от ошибок, связанных с использованием процедур, не предназначенных для обработки данных, связанных с этими переменными.

Наследование – это ещё одно базовое понятие объектно-ориентированного программирования. Наследование позволяет определять новые объекты, используя свойства прежних, дополняя или изменяя их. Объект-наследник получает все поля и методы «родителя», к которым он может добавить свои собственные поля и методы или заменить («перекрыть») их своими методами. Привлекательность наследования заключается в том, что если некий объект был уже определён и отлажен, он может быть использован и в других программах. При этом может оказаться, что новая задача отличается от предыдущей, и возникает необходимость некоторой модификации как данных, так и методов их обработки. Программисту приходится решать дилемму – создать объекты заново или использовать результаты предыдущей работы, применяя механизм наследования. Первый путь менее эффективен, так как требует дополнительных затрат времени на отладку и тестирование. Во втором случае часть этой работы оказывается выполненной, что сокращает время на разработку новой программы. Программист при этом может и не знать

деталей реализации объекта-родителя. Наследование позволяет создавать иерархические, связанные отношениями подчинения, структуры данных.

Виртуальный метод (виртуальная функция) – в объектно-ориентированном программировании метод (функция) класса, который может быть переопределён в классах-наследниках так, что конкретная реализация метода для вызова будет определяться во время исполнения. Таким образом, программисту необязательно знать точный тип объекта для работы с ним через виртуальные методы: достаточно лишь знать, что объект принадлежит классу или наследнику класса, в котором метод объявлен.

Виртуальные методы – один из важнейших приёмов реализации полиморфизма. Они позволяют создавать общий код, который может работать как с объектами базового класса, так и с объектами любого его класса-наследника. При этом базовый класс определяет способ работы с объектами и любые его наследники могут предоставлять конкретную реализацию этого способа.

Переменные объектного типа могут быть динамическими, то есть размещаться в памяти только во время их использования. Для работы с динамическими объектами используются расширенный синтаксис процедур New и Delete. Обе процедуры в этом случае содержат в качестве второго параметра вызов конструктора или деструктора для выделения или освобождения памяти переменной объектного типа.

Полиморфизм заключается в том, что одно и то же имя может соответствовать различным действиям в зависимости от типа объекта. Полиморфизм напрямую связан с механизмом позднего связывания. Решение о том, какая операция должна быть выполнена в конкретной ситуации, принимается во время выполнения программы. Когда существующая операция, например = или +, наделяется возможностью совершать действия над операндами нового типа, говорят, что такая операция является перегруженной. Перегрузка представляет собой частный случай полиморфизма и является важным инструментом ООП.

Конструктор – функция-член класса, определяющая способ создания объекта. Имя конструктора должно совпадать с именем класса.

Существует несколько форм конструктора. Наиболее часто используется конструктор с параметрами. Эти параметры используются для одновременной инициализации создаваемых объектов.

Деструктор – специальная функция класса, которая отвечает за уничтожение объекта. Имя деструктора совпадает с именем класса, перед которым ставится символ «тильда»: ~ .

Типичная программа на языке C++ состоит из совокупности объектов, взаимодействующих между собой посредством вызова методов друг друга. Структура программы на C++ приводится на рисунке 2.1.

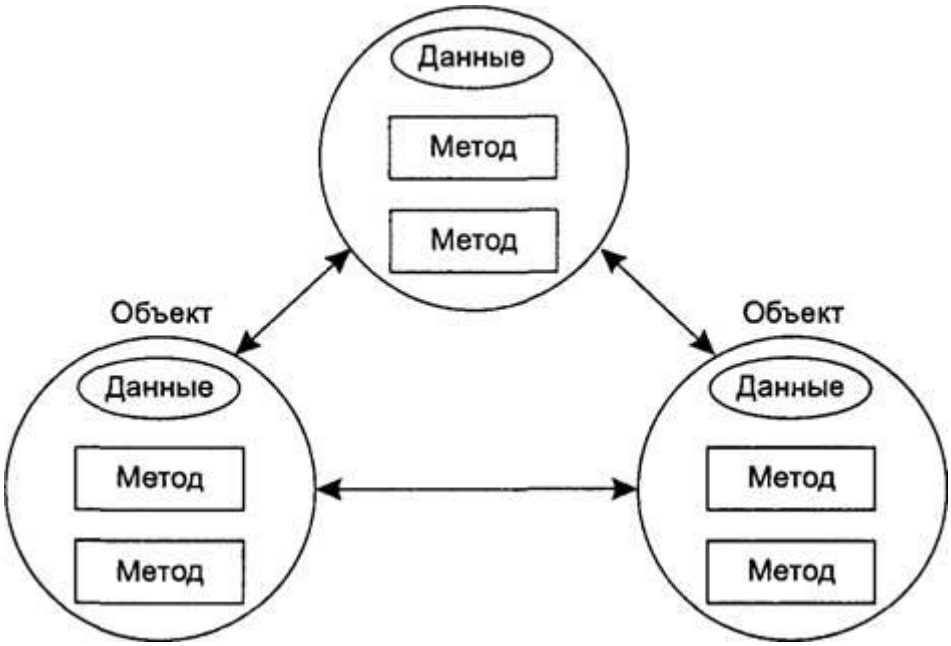

Рисунок 2.1 – Структура программы

2.2 Выбор инструментальных средств

Для построения моделей системы с рассмотрением различных её свойств целесообразно использовать язык разметки UML. В качестве программы, осуществляющей построение UML-диаграмм выбран Visual Paradigm.

В качестве IDE для разработки программы была выбрана Miscrosoft Visual Studio 2019, поскольку данная система программирования обладает следующими необходимыми функциями:

– подсветка синтаксиса;

– автодополнение кода;

– автоматическое выявление синтаксических и семантических ошибок;

– возможность использования объектно-ориентированного подхода;

– удобный интерфейс;

– наличие предкомпилированных модулей и библиотек.

2.3 Описание входных и выходных данных

Данные, фигурирующие в системе, можно разделить на входные, выходные и промежуточные. Также на эти данные накладываются определённые ограничения, возникающие вследствие технических характеристик компьютерной техники, представления данных в компьютере или вызванные непосредственно архитектурой и функционалом самой системы.

Входными данными является любая информация, поступающая в систему извне.

Выходные данные – это форматированные данные, выводимые программой в каком-либо виде (в файл, на устройство вывода и т.п.).

Промежуточные данные – набор данных, существующих лишь во время работы системы и использующиеся только системой на стадии обработки входных и формирования выходных данных.

Для данной предметной области перечень входных, выходных и промежуточных данных приведен в таблице 2.1.

Таблица 2.1 – Входные и выходные данные

Данные	Составляющие
Входные данные	1) номер команды – целочисленное число. Ограничение – 1 цифра, которая принадлежит диапозону чисел от нуля до максимального номера команды; 2) свойства объектов – строка или целочисленное число. Ограничение – перечисление нескольких значений через запятую.
Выходные данные	1) нумерованные команды и инструкции – строка, если команда, то с номером в начале.; 2) данные объектов, сведения – строка; 3) файлы cinemas.db, films.db, reperoires.db, содержащие структурированные строки с данными об объектах.
Промежуточные данные	Данные объектов, которые добавляли, удаляли или изменяли, существуют лишь во время работы системы до тех пор, пока не выйти из нее с помощью инструкций, иначе все измененные данные будут утеряны.

3 МЕТОД РЕШЕНИЯ

3.1 Объектно-ориентированный анализ

Выполним объектно-ориентированный анализ предметной области. В результате анализа предметной области был составлен словарь предметной области приведенный в таблице 3.1.

Таблица 3.1 – Словарь предметной области

Существительное	Глагол	Прочее
Кинотеатр	Ввести	Тип
	Сохранить	Ключ
Фильм	Найти	Название
	Обнаружить	Вне
Репертуар	Закрыть	Киностудия
	Открыть	Жанр
Вывод	Показать	Операторы
	Добавить	Продюсеры
	Удалить	Актеры
	Изменить	
	Сравнить	Место
	Очистить	Зал
		Цена
	Считать данные	Дата
	Записать данные	
		Состояние
	Выразить	Адрес
	Печатать	Категория
		Сеансы

На основе словаря предметной области составим объектно-ориентированный словарь – таблица 3.2.

Таблица 3.2 – ОО словарь предметной области

Класс/Сущность	Свойство/Состояние	Метод/Функция
Кинотеатр (Cinema)	Ключ - id Название – name Адрес – address Категория – category Кол-во мест – places Кол-во залов – halls	Сохранить ключ -set_id() Ввести – input() Найти – find() Показать – show() Сохранить – save()

Продолжение таблицы 3.2

Класс/Сущность	Свойство/Состояние	Метод/Функция
Кинотеатр (Cinema)	Репертуары – reps Состояние – state	
Фильм (Film)	Ключ - id Название – name Продюсеры – producers Операторы – opers Жанры – genres Актеры - actors	Сохранить ключ -set_id() Ввести – input() Найти – find() Показать – show() Сохранить – save() Обнаружить – search()
Репертуар (Repertoire)	Ключ - id Дата – date Цена – price Кол-во свободных мест – free_places Фильм - film	Сохранить ключ -set_id() Ввести – input() Найти – find() Показать – show() Сохранить – save() Обнаружить – search()
Вывод (Output)	Вне (out)	Выразить – put() Печатать – print()

3.2 Объектно-ориентированное проектирование

3.2.1 Диаграмма вариантов использования

Диаграмма вариантов использования (Use Case Diagram) позволяет описать список операций, которые выполняет система. Действующее лицо представляет собой любую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует её функциональные возможности для достижения определённых целей или решения частных задач.

На рисунке 3.1 изображена диаграмма вариантов использования для системы «Кинотеатры». В качестве актёров выступают объекты системы: информационная модель кинотеатра, фильма и репертуара.

Действиями являются функции системы, такие как: загрузка данных, ввод запроса, для ответа которого производится нужная работа с данными объектов, поиск объектов по определенным свойствам, вывод полученных результатов, сохранение данных.

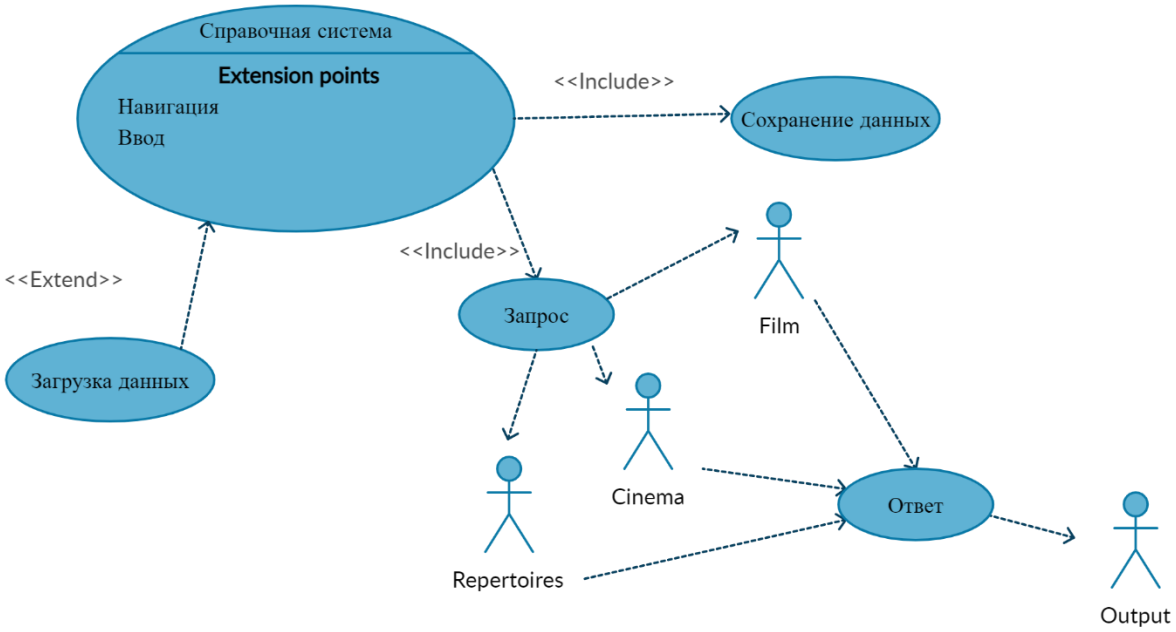

Рисунок 3.1 – Диаграмма вариантов использования

3.2.2 Диаграмма классов

Диаграмма классов (Class Diagram) служит для представления статической структуры модели систеы в терминах ООП. Диаграмма классов может отражать взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывать их внутреннюю структуру и типы отношений.

В качестве классов выделены актёры рассматриваемой системы – рисунок 3.2.

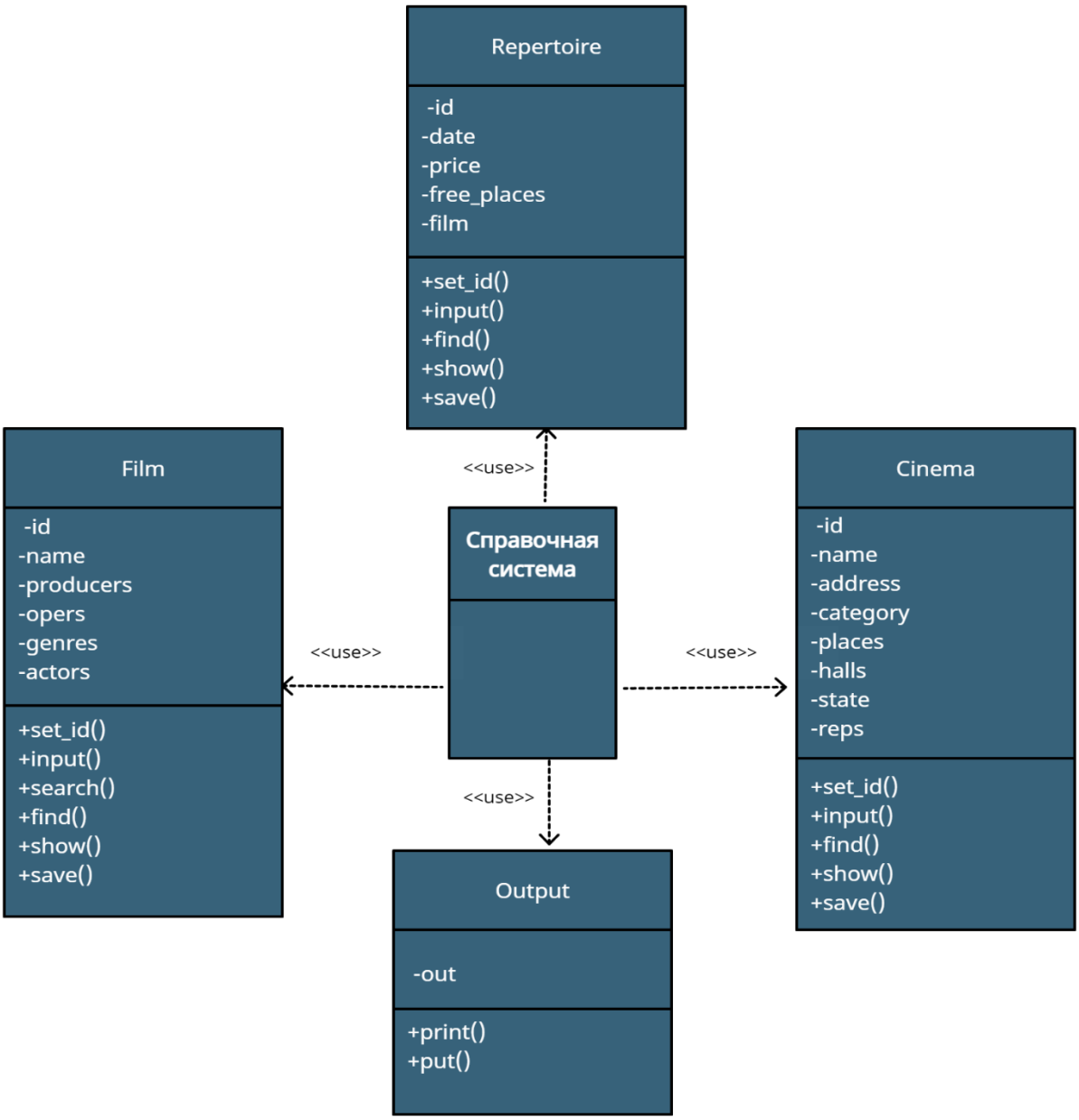

Рисунок 3.2 – Диаграмма классов

3.2.3 Диаграмма объектов

Диаграмма объектов строится на основе диаграммы классов, построенной ранее.

Данную диаграмму можно использовать для отображения одного из вариантов конфигурации объектов. Последний вариант очень полезен, когда допустимые связи между объектами могут быть сложными.

Объектам присваиваются уникальные имена, соответствующие своему назначению. При создании объектов инициализируются определённые свойства и выполняются определённые действия – рисунок 3.3.

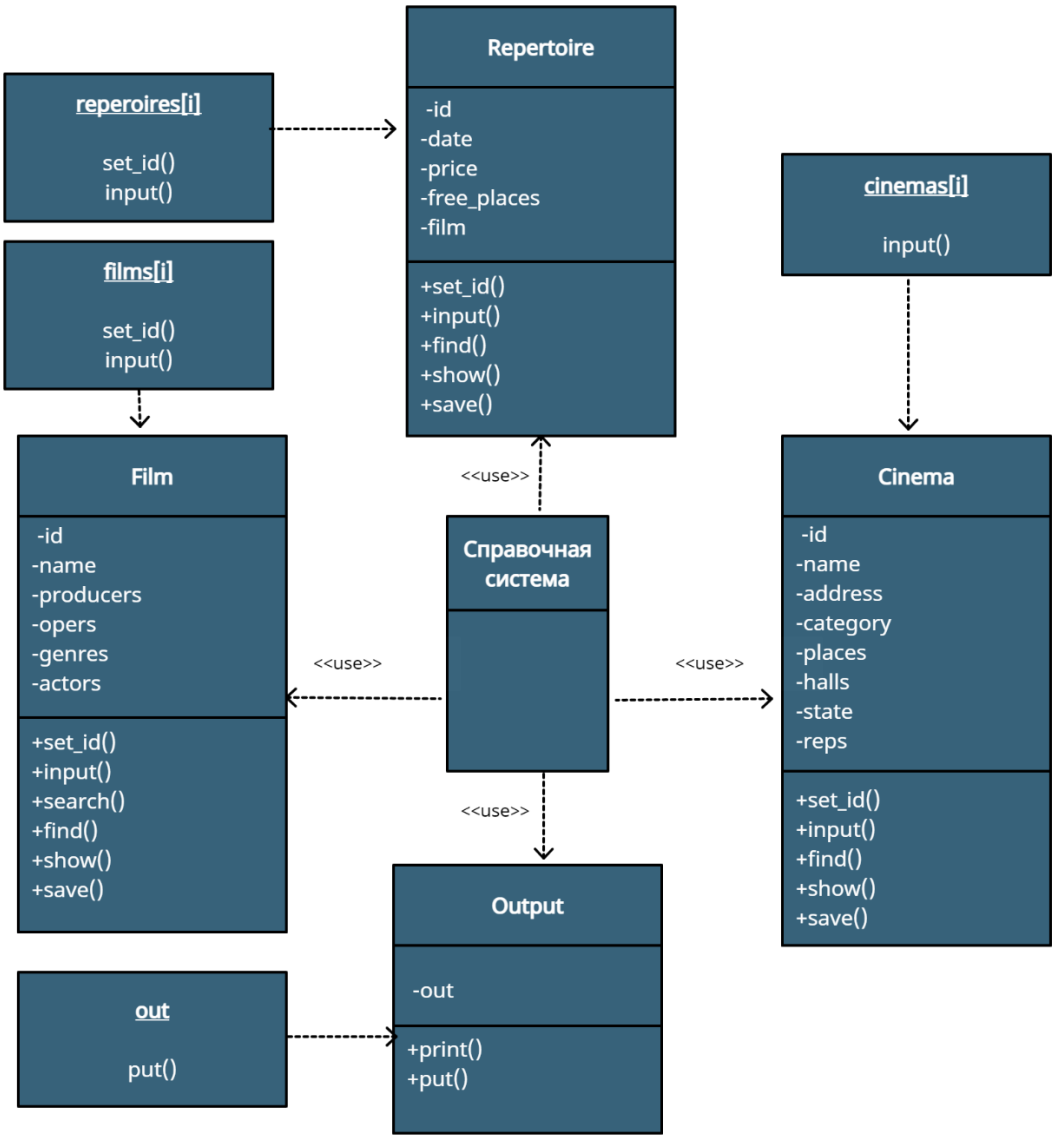

Рисунок 3.3 – Диаграмма объектов

3.2.4 Диаграмма состояний

Диаграмма состояний описывает всевозможные состояния системы и её частей во время работы. Переход от одного состояния к другому осуществляется при помощи специальных действий – методов, вызываемых в определённый момент времени выполнения программы.

Для данной предметной области было составлено 4 диаграммы приведенных на рисунках 3.4–3.7.

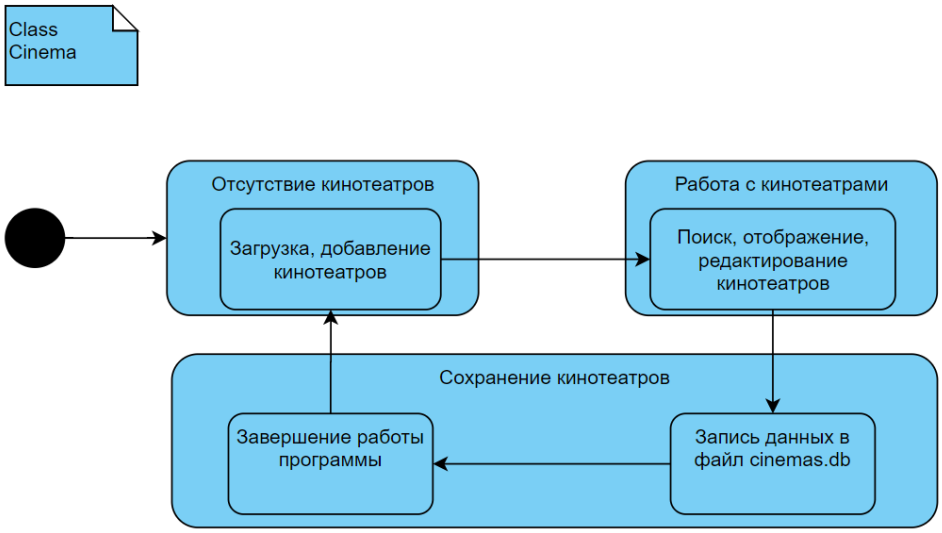

Рисунок 3.4 – Диаграмма состояний для класса Cinema (Кинотеатр)

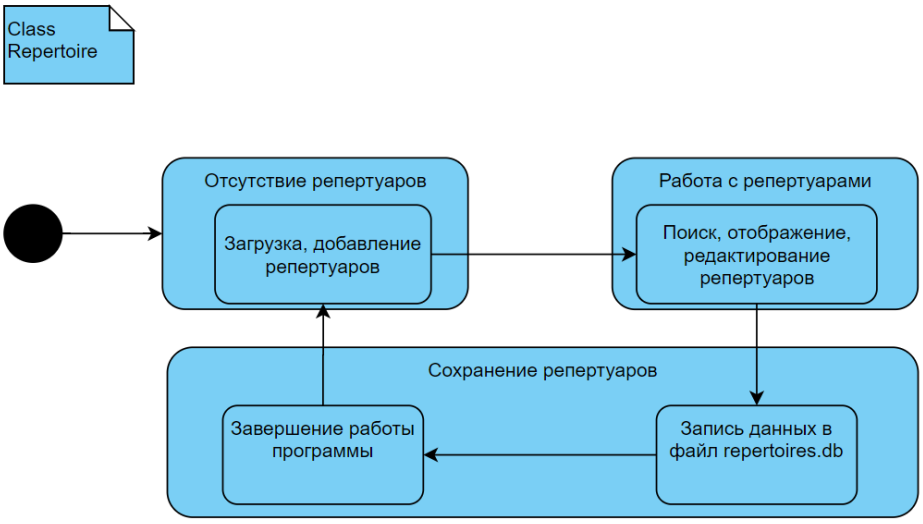

Рисунок 3.5 – Диаграмма состояний для класса Repertoire (Репертуар)

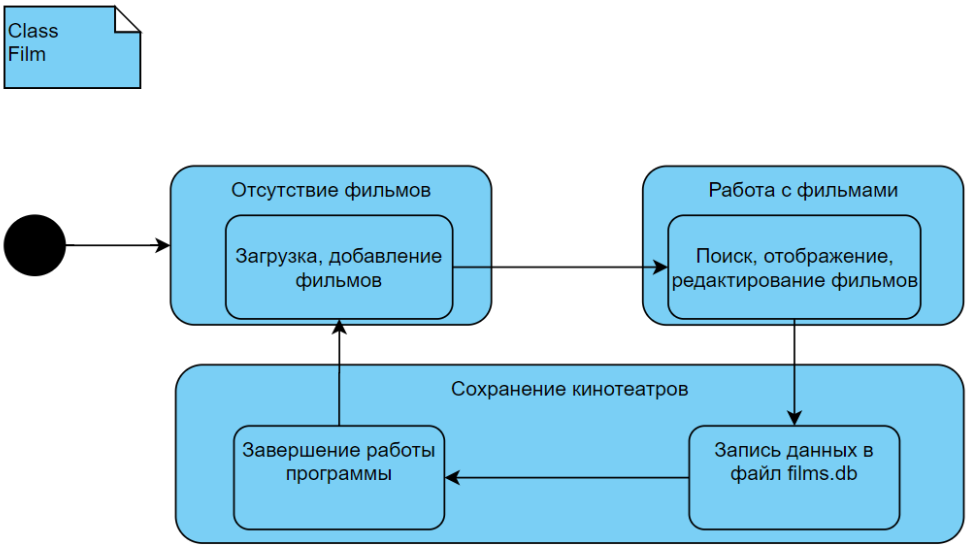

Рисунок 3.6 – Диаграмма состояний для класса Film (Фильм)

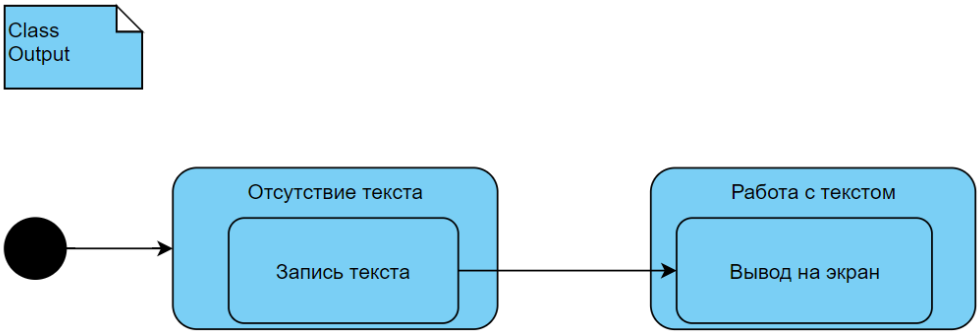

Рисунок 3.7 – Диаграмма состояний для класса Output (Вывод)

3.2.5 Диаграмма активности

При моделировании поведения проектируемой системы возникает необходимость не только представить процесс изменения её состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций.

Для этого используется диаграмма активности (Activity diagram) – она описывает алгоритмы вычислений или потоков управления в программных системах. Такая диаграмма изображена на рисунке 3.8.

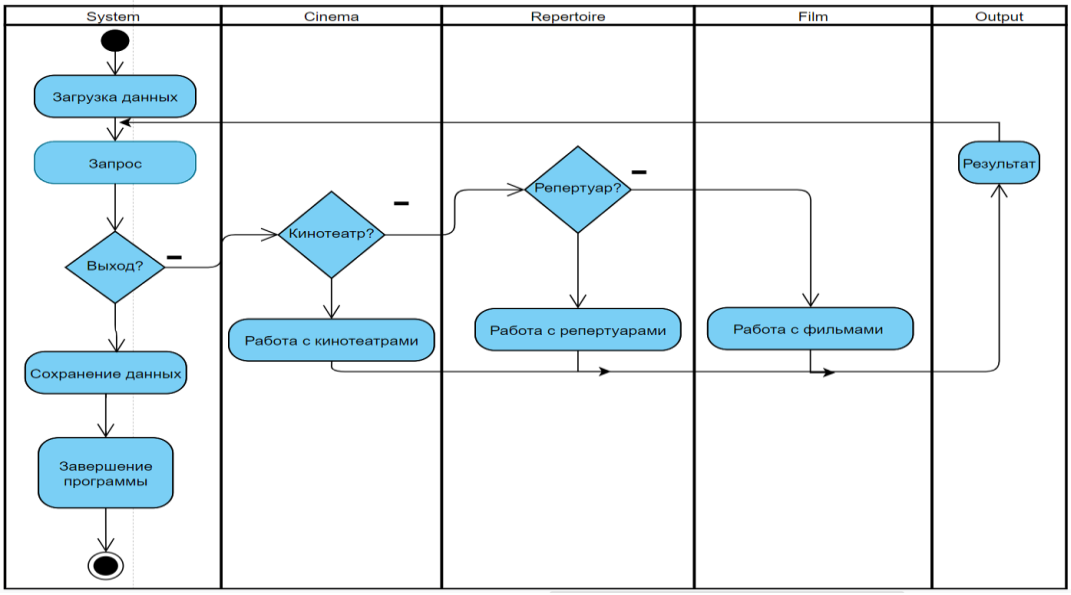

Рисунок 3.8 – Диаграмма активности

3.2.6 Диаграмма последовательности

Для представления временных особенностей передачи и приёма сообщений между объектами используется диаграмма последовательности. (Sequence diagram). На ней изображаются только те объекты, которые непосредственно участвуют во взаимодействии и не показываются возможные статические ассоциации с другими объектами – Рисунок 3.9.

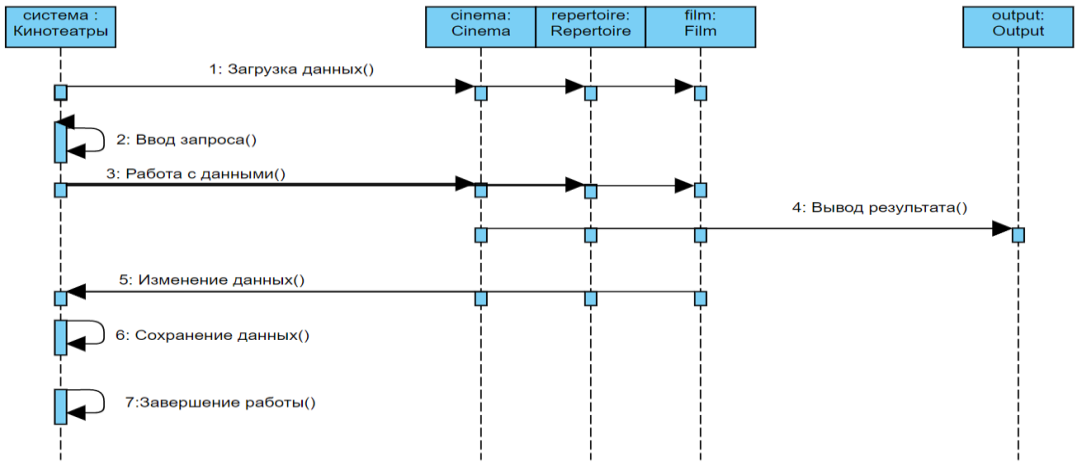

Рисунок 3.9 – Диаграмма последовательности

3.3 Объектно-ориентированное программирование

3.3.1 Диаграмма компонентов

Диаграмма компонентов описывает особенности физического представления системы. Она позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых выступает исходный, бинарный и исполняемый код – Рисунок 3.10.

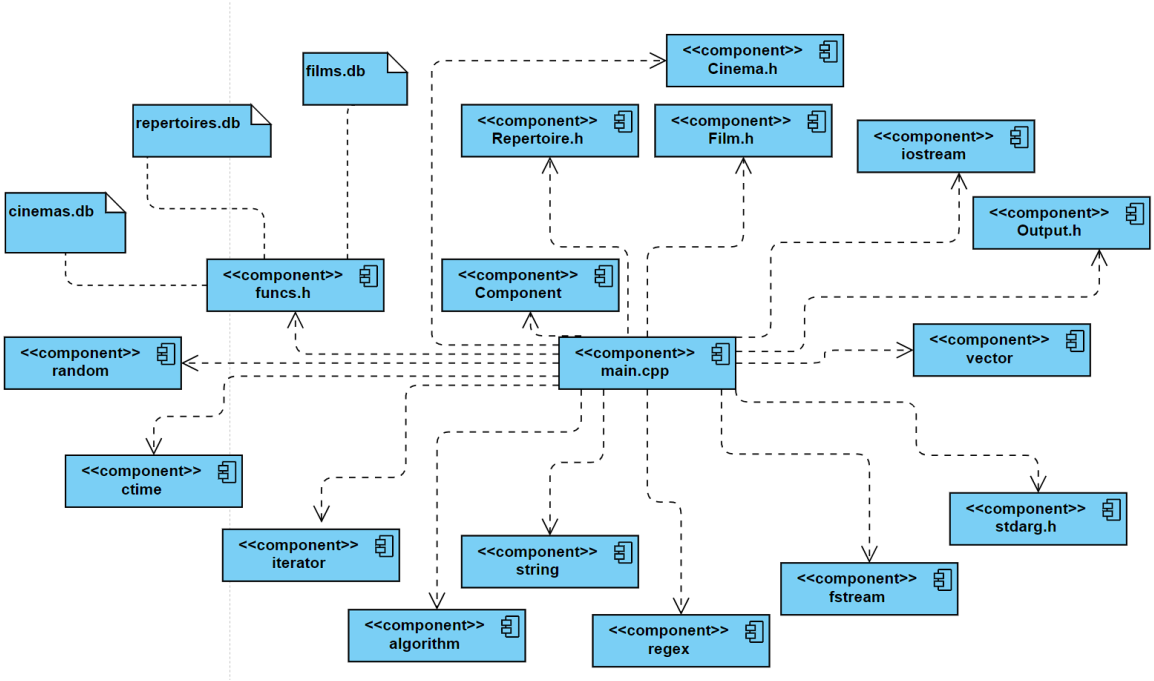

Рисунок 3.10 – Диаграмма компонентов

3.3.2 Диаграмма развёртывания

Диаграмма развёртывания (Deployment diagram) предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе её выполнения. Она содержит графические изображения процессов, устройств и связей между ними – Рисунок 3.11.

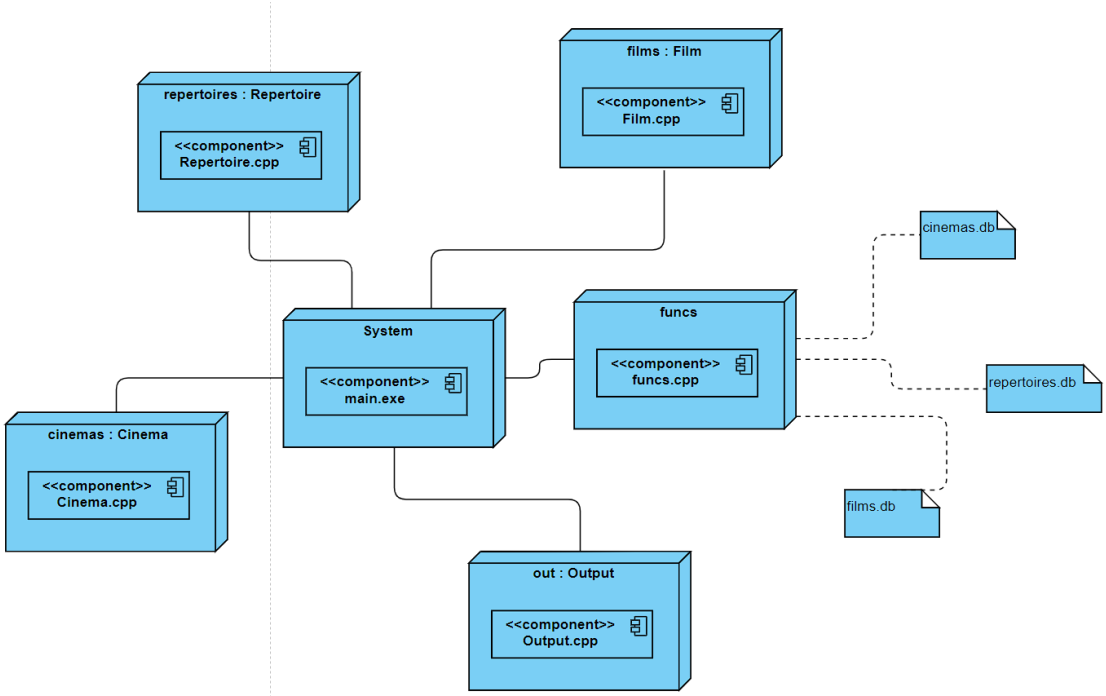

Рисунок 3.11 – Диаграмма развёртывания

3.3.3 Протоколы классов

Протокол класса определяет его допустимое поведение. В рамках предметной области было выделено 4 класса (Cinema, Output, Repertoire, Film).Протоколы этих классов приведены в таблице 3.3.

Таблица 3.3 – Протоколы классов

Класс	Свойства	Методы

Cinema	Public:
	int id;// ключ
	int places; //кол-во мест
	int halls; //кол-во залов
	string name; //название
	string address;//адрес
	string category;//категория
	bool state;//состояние
	vector<Repertoire *> reps;//репертуары
		Public:
		
		void set_id();//установка уникального ключа
		void input();//ввод нового кинотеатра
		void find();//поиск кинотеатров
		void show();//вывод сведений кинотеатра
		void save();//сохранение сведений в файл cinemas.db

Продолжение таблицы 3.1

Класс	Свойства	Методы

Output	Public:
	string out; //строка вывода
		Public:
		void put(string a);//вывод a
		void print();//вывод out

Repertoire	Public:
	int id; //ключ
	int price; //цена
	int free_places;//кол-во свободных мест
	string date;//дата
	Film *film;//фильм
		Public:
		Repertoire *search(int &j);//поиск репертуара по ключу
		void set_id();();//установка уникального ключа
		void input();//ввод нового репертуара
		void find();//поиск репертуаров
		void save();//сохранение сведений в файл repertoires.db
		void show();//вывод сведений репертуара

Film	Public:
	int id;//ключ
	string name;//название
	string studio;//название киностудии
	vector<string> producers;//продюсеры
	vector<string> opers;//операторы
	vector<string> genres;//жанры
	vector<string> actors;//актеры
		Public:
		Film *search(int &j);//поиск фильма по ключу
		void set_id();();//установка уникального ключа
		void input();//ввод нового фильма
		void find();//поиск фильмов
		void save();//сохранение сведений в файл films.db
		void show();//вывод сведений фильма

ЗАКЛЮЧЕНИЕ

В ходе работы была изучена предметная область «Кинотеатры», был произведен объектно-ориентированный анализ. На основании выделенных классов были построены UML-диаграммы (Диаграмма вариантов использования, диаграмма классов, диаграмма объектов, диаграмма состояний, диаграмма активности, диаграмма последовательности, диаграмма компонентов, диаграмма развёртывания), характеризующие структуру системы и происходящие в ней процессы.

Результатом работы является рабочая программа, написанная на языке C++. Её создание базируется на парадигме объектно-ориентированного программирования. Все классы четко определяют свои свойства и методы. Для облегчения написания и чтения кода проект состоит из множества заголовочных файлов, каждый из которых отвечает за определённую часть программы – класс или набор вспомогательных функций.

Произведена работа с текстовыми файлами – чтение и запись свойств объектов классов. Данные объектов хранятся в определённом формате, обработка производится согласно постановке задачи. Загрузка и сохранение данных происходит автоматически при выполнении инструкций требуемых программой.

Оптимизированный код обеспечивает быстродействие приложения и эффективное расходование системных ресурсов. В программе предусмотрены варианты неверного ввода пользователя, что обозначает высокую надежность и работоспособность системы. Разработан простой и интуитивно понятный интерфейс для минимизации возникновений трудностей с использованием системы.

Программа имеет широкое применение в своей области. Благодаря ей можно быстро и точно определить нужный клиенту фильм, кинотеатр, репертуар и эффективно провести оставшееся время.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шлеер, С. Объектно-ориентированный анализ: моделирование мира в состояниях / С. Шлеер, Меллор С. – К.: Диалектика, 1993. – 240 с.
2. Фаулер, М. UML. Основы / М. Фаулер, К. Скотт – СПб.: Символ-Плюс, 2002. – 2-е изд. – 192 с.
3. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Дж. Рамбо, А. Якобсон. – 2007. – 2-е изд. – 496 с.
4. Якобсон, А. Унифицированный процесс разработки программного обеспечения / А. Якобсон, Г. Буч, Дж. Рамбо – СПб.: Питер, 2002. – 496 с.
5. Арлоу, Д. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Д. Арлоу, И. Нейштадт – 2007. – 2-е изд. – 624 с Издательство Символ-Плюс.
6. Кватрани, Т. Rational Rose 2000 и UML. Визуальное моделирование / Т. Кватрани – М.: ДМК Пресс, 2001. – 176 с.
7. Мацяшек, Л. Анализ требований и проектирование систем. Разработка информационных систем с использованием UML / Лешек А. Мацяшек. – М.: Вильямс, 2002. – 432 с.
8. Леоненков, А.В. Самоучитель UML / А.В. Леоненков. – СПб.: БХВ, 2004. – 2-е изд. – 158 с.
9. Леоненков, А.В. Язык UML 2 в анализе и проектировании программных систем и бизнес-процессов [Электронный ресурс]. – Режим доступа: http://www.intuit.ru/department/se/uml2/
10. Страуструп, Б. Язык программирования C++. Специальное издание / Бьерн Страуструп. – М.: Бином, 2008. – 3-е изд. – 1054 с.
11. Холзнер, С. Учебный курс Visual C++ 6 / С. Холзнер– СПб.: Питер, 2007. – 570 с.
12. Horton, I. Beginning Visual C++ 2008 / Ivor Horton. – Indianapolis: Wiley, 2008. – 1394 p.
13. Пахомов, Б.И. C/C++ и MS Visual C++ 2008 для начинающих / Б.И. Пахомов – СПб.: БХВ-Петербург, 2009. – 624 с.
14. Норкене, Е.А. Методические указания по оформлению студенческих работ / Е.А. Норкене, Н.П. Пулинец – Донецк: ДонНТУ, 2017. – 32 с.

ПРИЛОЖЕНИЕ А

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

А.1. Общие сведения

Тема курсового проекта: «Имитационное моделирование динамических систем и процессов с использованием объектно-ориентированного подхода. Кинотеатры».

Система проектируется студентом 1-го курса ДонНТУ факультета КНТ, группы ПИ-20Г, Евсеевым Максимом Алексеевичем.

Основанием для разработки программного продукта является задание, выданное кафедрой ИИСА. Плановый срок начала работы по созданию системы имитационного моделирования: 12.02.2021, срок окончания: 20.05.2021. Курсовой проект должен выполняться согласно графику, приведенному в таблице А.1.

Таблица А.1 – Этапы, результаты и сроки разработки ПП

№	Этап работы	Результат работы	Срок выполнения (№ недели)
1	Получение задания на КП	Задание на разработку (1 стр.)	1-2
2	Выявление требований К разрабатываемому Программному продукту	Техническое задание (3-5 стр.)	2-3

Продолжение таблицы А.1

№	Этап работы	Результат работы	Срок выполнения (№ недели)
3	Проведение ОО анализа предметной области	Словарь предметной Области. Сценарии Использования системы. ОО словарь предметной области.	3-4
4	Проведение ОО Проектирования	Диаграммы классов объектов	5-6
5	Проведение ОО Проектирования	Диаграммы состояний и переходов, взаимодействия	7-8
6	Проектирование протоколов классов	Протоколы классов	8-9
7	Реализация классов	Описание реализации классов	9-10
8	Реализация и отладка программы. Проведение тестирования ПП.	Текст программы. Описание программы и тестов.	11-12
9	Проведение имитационного моделирования, получение статистики работы.	Экранные формы(1 -2 стр.). Руководство Пользователя (1 стр.)	11-12

Продолжение таблицы А.1

№	Этап работы	Результат работы	Срок выполнения (№ недели)
10	Оформление пояснительной записки и сопроводительных материалов.	Прошитая ПЗ с CD-ROM (30-50 стр.),сдается Преподавателю лично не позже чем за 3 дня до защиты КП	13
11	Защита курсового Проекта		14

А.2 Назначения и цели создания программы

Данный программный продукт предназначен для имитации работы динамических объектов реального мира – кинотеатры. Получение статистических данных: сведения о кинотеатрах города, о фильмах, которые в них демонстрируются, о сеансах и билетах на эти сеансы.

А.3 Характеристика объекта автоматизации

Предметной областью курсового проекта является кинотеатры. Требуется разработать программную систему, предназначенную для работников справочной службы кинотеатров города. Такая система должна обеспечивать хранение сведений о кинотеатрах города, о фильмах, которые в них демонстрируются, о сеансах и билетах на эти сеансы. На разных сеансах в одном кинотеатре могут идти разные фильмы, а если в кинотеатре несколько залов, то и на одном.

Таблица А.2 – Ключевые поля объектов автоматизации

Кинотеатр	Фильм	Репертуар
Название Адрес Категория Количество мест Количество залов Состояние	Название Режиссер Оператор Список актеров Жанр Киностудия	Дата Сеанс Цена Количество свободных мест

Должны быть созданы обобщенные списки:

– сведения о кинотеатрах;

– сведения о фильмах;

– сведения о репертуаре.

Справочной службе могут потребоваться следующие сведения о текущем состоянии проката фильмов в городе:

– репертуар кинотеатра;

– в каких кинотеатрах можно посмотреть боевики;

– число свободных мест на заданный сеанс в указанном кинотеатре;

– цена билетов на заданный сеанс в указанном кинотеатре;

– какие фильмы заданного режиссера демонстрируются в кинотеатрах;

– в каких кинотеатрах демонстрируются комедии.

Должна быть предусмотрена возможность добавления и удаления фильма.

А.4 Требования к программному продукту

А.4.1 Требования к системе в целом

В целом к системе предъявляются следующие требования:

а) имитация работы работников справочной службы;

б) ввод, сохранение и вывод обобщенных списков:

– сведения о кинотеатрах:

– сведения о фильмах;

– сведения о репертуаре.

А.4.2 Требования к задачам и функциям программного продукта

В процессе работы необходимо обеспечить выполнение следующих функций:

а) ввод и вывод сведений о кинотеатрах, фильмах, репертуаре на экран;

б) сохранение в файл и загрузка из файла информации о кинотеатрах;

в) поиск определенных сеансов по заданным критериям (жанр, режиссер, актеры и т.д.)

г) отображение текущих кинотеатров, фильмов и репертуаров;

д) редактирование данных объектов и последующее сохранение;

е) возможность выхода из приложения и отмены текущего действия.

А.4.3 Требования к техническому обеспечению

К техническому обеспечению предъявляются следующие требования:

а) процессор – двухъядерный (уровня Pentium 4 и выше);

б) объем оперативной памяти – не менее 2Гб;

в) свободное дисковое пространство – около 100 Мб. Не менее 50 Мб свободного дискового пространства для временных файлов;

г) графический адаптер – VGA-совместимый;

д) монитор –VGA-совместимый;

е) клавиатура.

А.4.4 Требования к программному обеспечению

Для стабильной работы к программному обеспечению предъявляется следующее требования:

а) обеспечить удобный и понятный пользовательский интерфейс;

б) реализовать программу в виде отдельных классов;

в) организовать защиту от некорректного ввода начальных параметров;

г) обеспечить надежное хранение информации.

Программным обеспечением для проектирования программы является CASE-средство Microsoft Visio ,а для разработки – Microsoft Visual Studio 2019. Для запуска программы необходимо наличие операционной системы Windows 7,8,10 и соответствующих библиотек Microsoft Visual C++ Redistributable.

А.4.5 Требования к организационному обеспечению

В программную документацию должны входить:

а) пояснительная записка;

б) приложения:

- техническое задание;
- руководство пользователя;
- экранные формы;
- листинг программы.

А.4.6 Требования к комплекту поставки ПП

Программный продукт поставляется с пояснительной запиской к курсовому проекту в файле ПЗ.doc, руководством пользователя в файле Help.doc, исходными кодами в виде проекта среды разработки в папке SRC, исполнительным файлом программы main.exe и файлом с данными о разработчике readme.txt.

ПРИЛОЖЕНИЕ Б

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для корректной работы программы необходимо следовать следующим шагам:

1) запустите программу через файл main.exe;

2) следуя инструкциям выведенным на экране, выберите область с которой вам нужно работать и введите ее номер;

3) после выбора выберите действие с данным типом объекта и введите его номер, после чего следуйте инструкциям;

4) чтобы вернуться или выйти из программы введите число 0;

5) чтобы отменить ввод свойств введите число -1;

6) чтобы выйти из программы следуйте инструкциям, иначе потеряются несохраненные данные.

ПРИЛОЖЕНИЕ В

ЭКРАННЫЕ ФОРМЫ

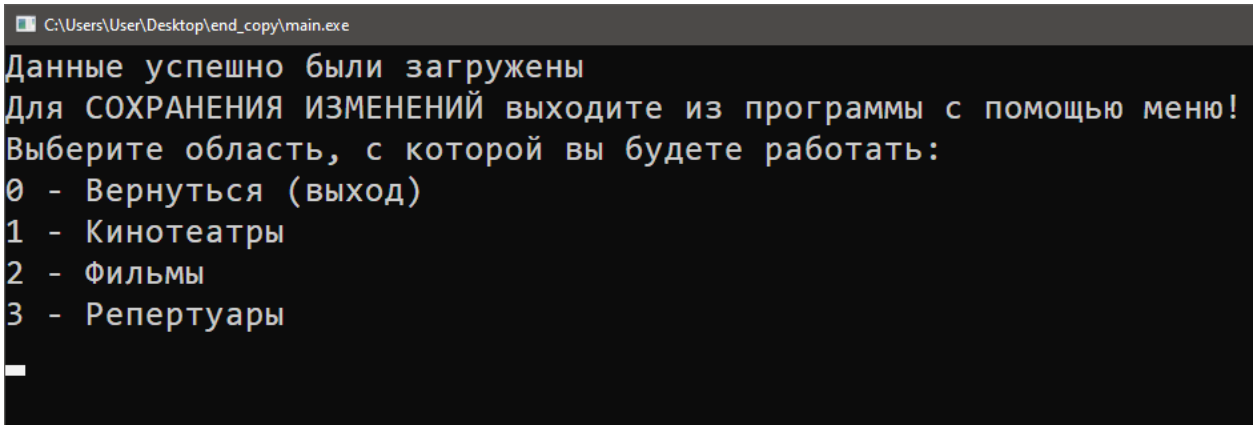

Рисунок В.1 – Запуск программы main.exe

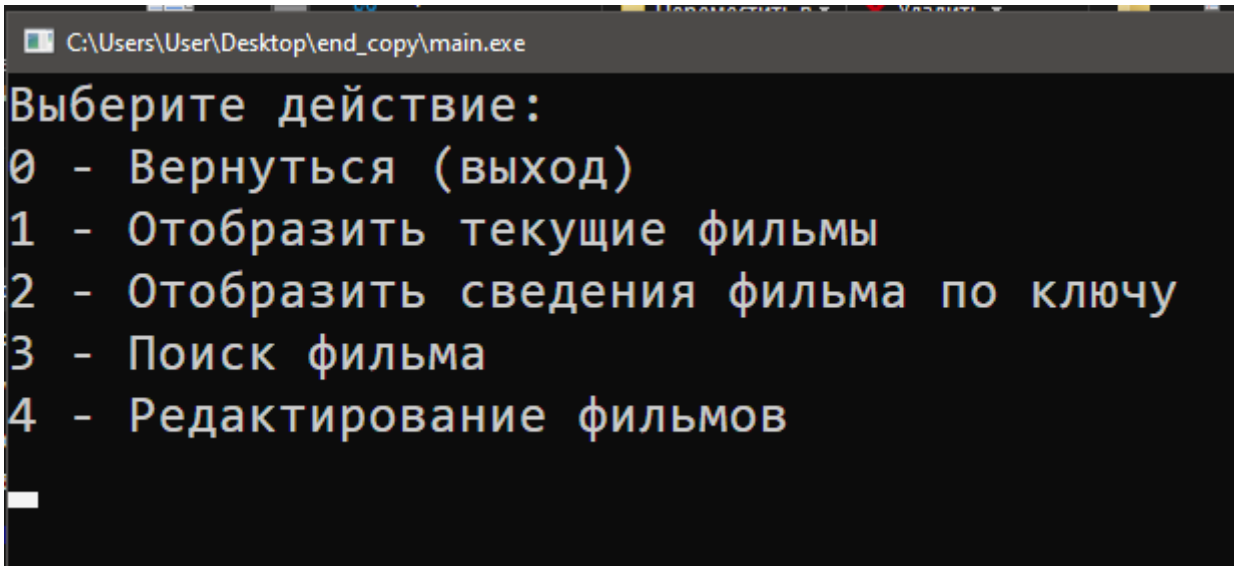

Рисунок В.2 – Выборка действий после ввода “2”

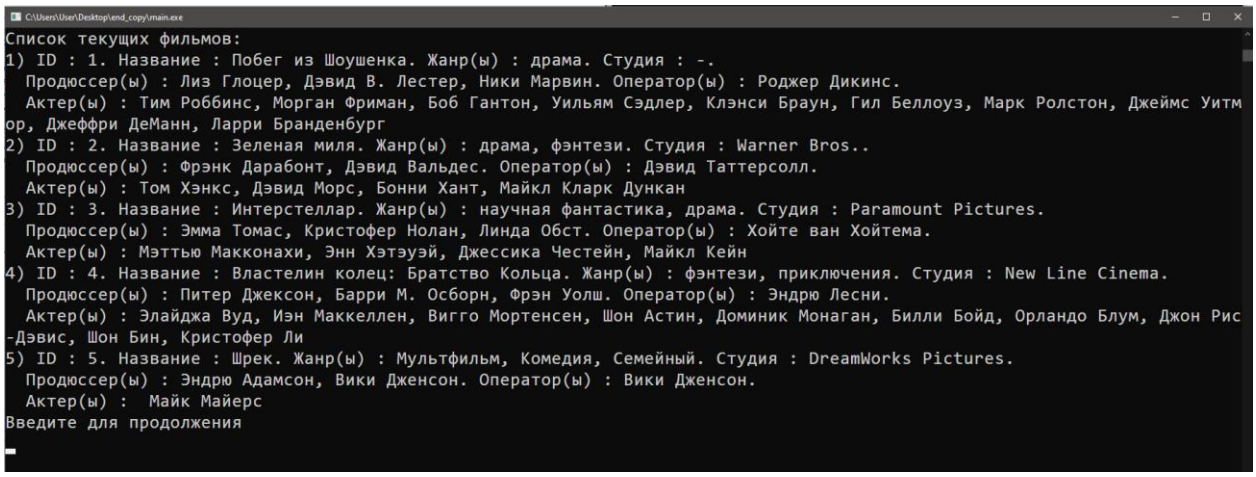

Рисунок В.3 – Действие: Отобразить текущие фильмы

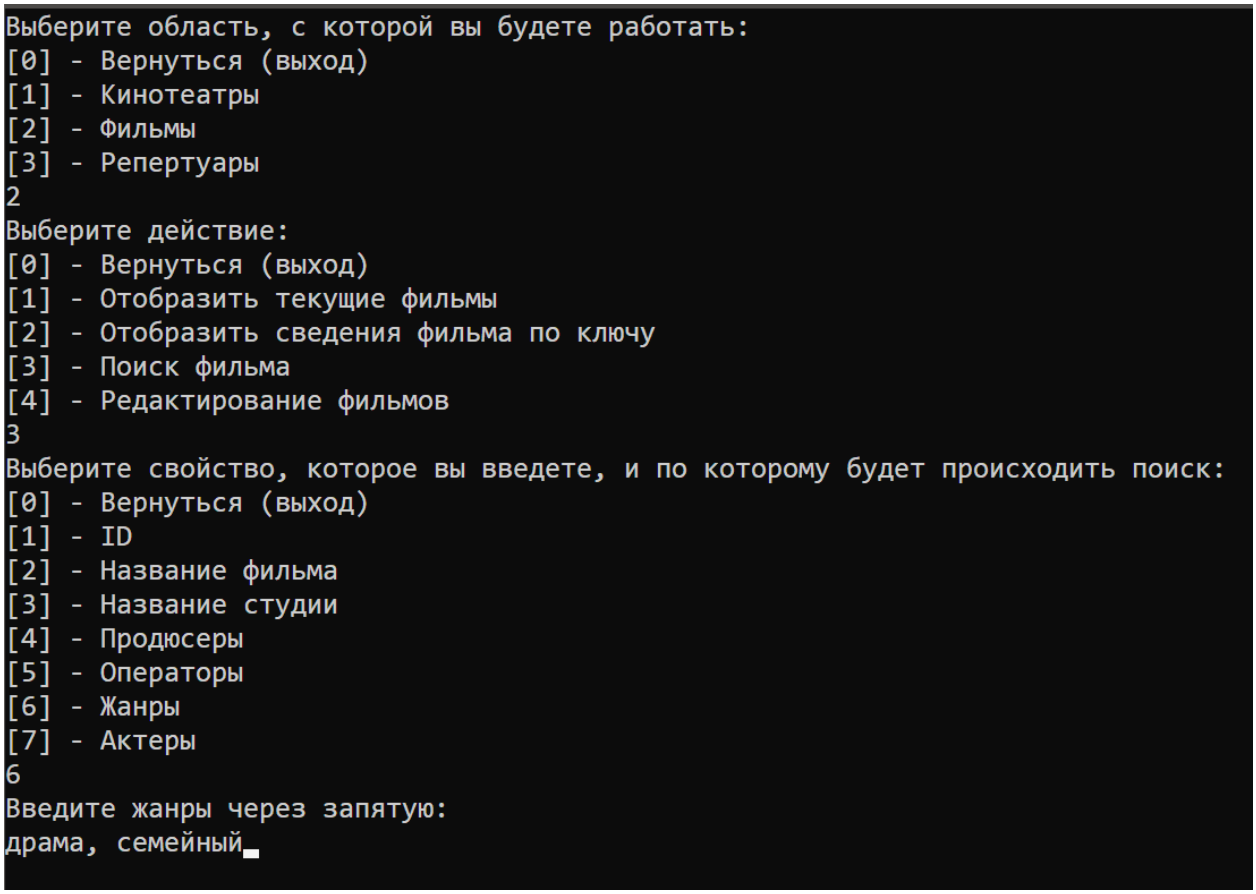

Рисунок В.5 – Действие: Поиск фильма по жанрам

```
драма, семейный

ID : 1. Название : Побег из Шоушенка. Жанр(ы) : драма. Студия : -.
Продюссер(ы) : Лиз Глоцер, Дэвид В. Лестер, Ники Марвин. Оператор(ы) : Роджер Дикинс.
Актер(ы) : Тим Роббинс, Морган Фриман, Боб Гантон, Уильям Сэдлер, Клэнси Браун, Гил Беллоуз, Марк Ролстон, Джеймс Уитмор
, Джеффри ДеМанн, Ларри Бранденбург

ID : 2. Название : Зеленая миля. Жанр(ы) : драма, фэнтези. Студия : Warner Bros..
Продюссер(ы) : Фрэнк Дарабонт, Дэвид Вальдес. Оператор(ы) : Дэвид Таттерсолл.
Актер(ы) : Том Хэнкс, Дэвид Морс, Бонни Хант, Майкл Кларк Дункан

ID : 3. Название : Интерстеллар. Жанр(ы) : научная фантастика, драма. Студия : Paramount Pictures.
Продюссер(ы) : Эмма Томас, Кристофер Нолан, Линда Обст. Оператор(ы) : Хойте ван Хойтема.
Актер(ы) : Мэттью Макконахи, Энн Хэтэуэй, Джессика Честейн, Майкл Кейн

ID : 5. Название : Шрек. Жанр(ы) : Комедия, Семейный, Мультфильм. Студия : DreamWorks.
Продюссер(ы) : -. Оператор(ы) : -.
Актер(ы) : Зеленый огр

Введите для продолжения:
```

Рисунок В.6 – Результат поиск фильма по жанрам

```
C:\Users\User\Desktop\end_copy\main.exe
Список текущих кинотетров:
1)   ID : 1. Название : Звездочка. Адрес : Университетская ул. 57, Донецк. Категория : все.
  Кол-во мест : 280. Кол-во залов : 4. Состояние : Работает.
        Репертуары :

         1) ID : 1. Дата : 25.05.2021. Цена : 175.
            Кол-во свободных мест: : 60.
            Фильм :

                    ID : 4. Название : Властелин колец: Братство Кольца. Жанр(ы) : фэнтези, приключения. Студия : New
Line Cinema.
                    Продюссер(ы) : Питер Джексон, Барри М. Осборн, Фрэн Уолш. Оператор(ы) : Эндрю Лесни.
                    Актер(ы) : Элайджа Вуд, Иэн Маккеллен, Вигго Мортенсен, Шон Астин, Доминик Монаган, Билли Бойд, Ор
ландо Блум, Джон Рис-Дэвис, Шон Бин, Кристофер Ли
_____

Введите для продолжения
```

Рисунок В.7 – Действие: Отобразить текущие кинотеатры

```
C:\Users\User\Desktop\end_copy\main.exe
Для СОХРАНЕНИЯ ИЗМЕНЕНИЙ выходите из программы с помощью меню!
Выберите область, с которой вы будете работать:
0 - Вернуться (выход)
1 - Кинотеатры
2 - Фильмы
3 - Репертуары
3
Выберите действие:
0 - Вернуться (выход)
1 - Отобразить текущие репертуары
2 - Отобразить сведения репертуара по ключу
3 - Поиск репертуара
4 - Редактирование репертуаров
4
Выберите действие:
0 - Вернуться (выход)
1 - Добавить реперутар
2 - Удалить реперутар
3 - Изменить реперутар
1
Введите дату в формате ДДММГГ (например, 30052021):
26052021
Введите ID фильма
4
Введите цену билета
200
Введите кол-во свободных мест
70
Успех
```

Рисунок В.8 – Добавление репертуара

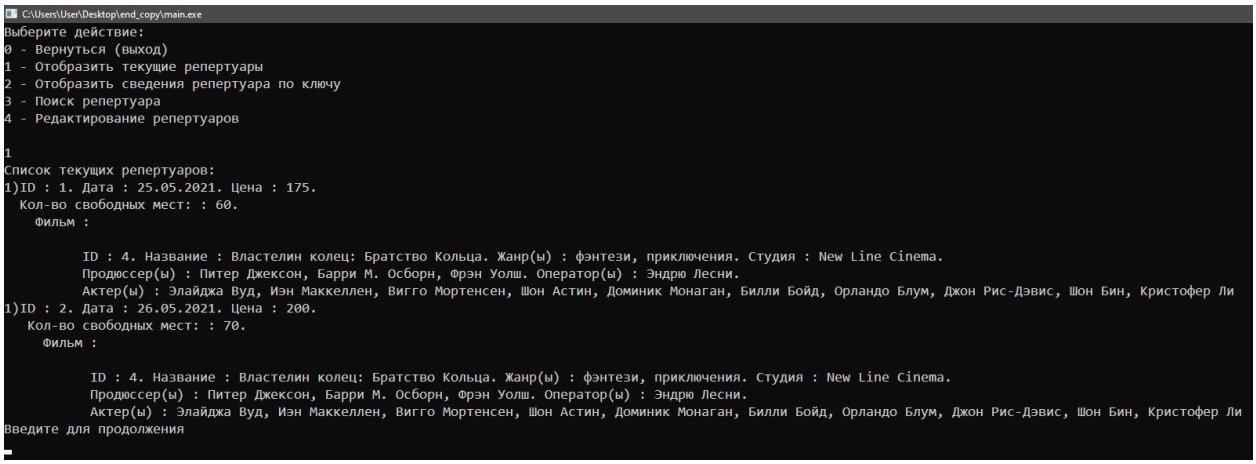

Рисунок В.9– Отображение текущих репертуаров

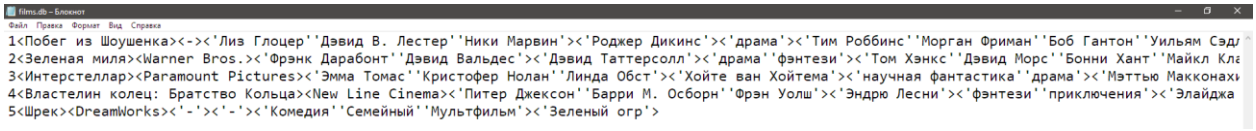

Рисунок В.10 – Файл films.db

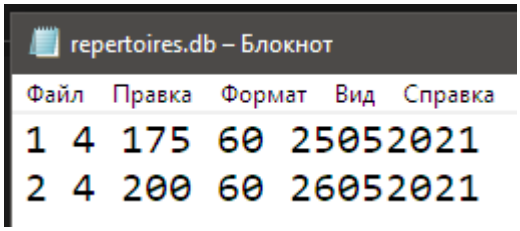

Рисунок В.11 – Файл repertories.db

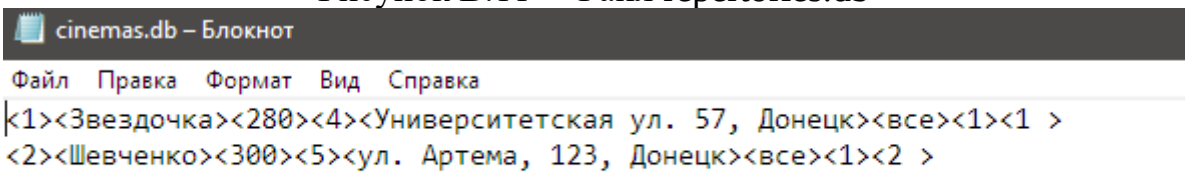

Рисунок В.12 – Файл cinemas.db

ПРИЛОЖЕНИЕ Г

ЛИСТИНГ ПРОГРАММНЫХ МОДУЛЕЙ

Г.1 Файл main.cpp

```
#include <iostream>
#include <vector>
#include <stdarg.h>
#include <fstream>
#include <string>
#include <regex>
#include <algorithm>
#include <iterator>
#include <ctime>
#include <random>

#include "Output.h"
#include "Film.h"
#include "Cinema.h"
#include "Repertoire.h"
#include "funcs.h"
#define d(a) a[0] << a[1] << "." <<
a[2] << a[3] << "." << a.substr(4,
4)
#define rand(x) engine() % x
#define ctoi(j)
atoi(string({j[0]}).c_str())
using namespace std;

vector<Film> films;
vector<Cinema> cinemas;
vector<Repertoire> repertoires;

int main()
{
    std::mt19937 engine;
    engine.seed(std::time(nullptr));
    system("chcp 1251");
    cls();
    Output out;
    int x, y;
    get_all();
    for (;;)
    {
        out.put("Для СОХРАНЕНИЯ
ИЗМЕНЕНИЙ выходите из программы с
помощью меню!");
        cinx(y, "Выберите область, с
которой вы будете работать:", 3,
"Кинотеатры", "Фильмы",
"Репертуары");
        switch (y)
        {
        case 1:
        {
            for (;;)
            {

                out.put("Нажата 1");
                cinx(x, "Выберите
действие:", 4, "Отобразить текущие
кинотеатры", "Отобразить сведения
кинотеатра по ключу",
                    "Поиск
кинотеатров", "Редактирование
кинотеатров");
                switch (x)
                {
                case 1:
                {
                    out.put("Список
текущих кинотетров:");
                    for (Cinema &a :
cinemas)
                        a.show();
                    out.put("Введите
для продолжения");
                    getchar();
                    cls();
                    break;
                }
                case 2:
                {
                    int j;
                    out.put("Введите
ключ нужного вам кинотеатра или -1
для выхода");
                    r_cin(j);
                    if (j == -1)
                    {
                        cls();
                        break;
                    }
                    for (Cinema &a :
cinemas)
                    {
                        if (j ==
a.id)
                        {
```

```
a.show();

out.put("Введите для продолжения:");
                                cc();

getchar();
                                cls();
                                j = -1;
                            }
                        }

                        if (j != -1)
                        {

out.put("Кинотеатра с данным ключом
не существует");

out.put("Введите для продолжения:");
                            cc();
                            getchar();
                            cls();
                        }
                        break;
                    }
                    case 3:
                    {
                        int j, i;
                        string b;
                        Cinema a(0);

                        cinx(j,
"Выберите свойство, которое вы
введете, и по которому будет
происходить поиск:",
                             8, "ID",
"Название кинотеатра", "Кол-во
залов", "Кол-во мест", "Адрес",

"Категория", "Состояние", "ID
репертуаров");
                        if (j == 0)
                        {
                            cls();
                            break;
                        }

                        if (j == 1)
                        {

out.put("Введите ID:");
                            r_cin(a.id);
                        }

                        if (j == 2)
                        {

out.put("Введите название
кинотеатра:");


push_line(a.name);
                        }

                        if (j == 3)
                        {

out.put("Введите кол-во залов:");

r_cin(a.halls);
                        }

                        if (j == 4)
                        {

out.put("Введите кол-во мест");

r_cin(a.places);
                        }
                        if (j == 5)
                        {

out.put("Введите адрес:");

push_line(a.address);
                        }
                        if (j == 6)
                        {

out.put("Введите категорию:");

push_line(a.category);
                        }
                        if (j == 7)
                        {

out.put("Введите состояние:");
                            r_cin(i);
                            a.state = (i
== 1);
                        }
                        if (j == 8)
                        {

out.put("Введите ID репертуаров
через Enter или -1 для выхода:");

push_vector(a.reps);
                        }
                        cout << endl;
                        a.find();

                        break;
                    }
```

```
                case 4:
                {
                    out.put("Нажата
4");
                    cinx(x,
"Выберите действие:", 3, "Добавить
кинотеатр", "Удалить кинотеатр",
"Изменить кинотеатр");

                    switch (x)
                    {
                    case 1:
                    {

cinemas.push_back(Cinema());

out.put("Успешно добавлен");

out.put("Введите для продолжения:");
                        getchar();
                        cls();
                        break;
                    }

                    case 2:
                    {
                        for (;;)
                        {
                            int j;
                            int x =
rand(10);
                            cinx(j,
"Введите ключ фильма, который вы
хотите удалить или -1 для выхода:");
                            if (j ==
-1)
                            {

cls();

break;

                            }

                            for
(auto it = films.begin(); it <
films.end(); it++)
                            {
                                if
((*it).id == j)
                                {


out.put("Фильм который вы хотите
удалить:");

(*it).show();

cout << "Для подтверждения удаления
введите " << x << " :";

cinx(j, "");

if (j == x)

{

films.erase(it);

out.put("Успех");

j = -1337;

}

else

{

out.put("Отмена");

j = -1337;

}

break;
                                }
                            }
                            if (j !=
-1337)
                            {

out.put("Фильм не был найден");
                            }
                        }

                        break;
                    }
                    case 3:
                    {
                        int j;
                        int x;
                        cinx(j,
"Введите ID фильма, который нужно
изменить:");
                        for (size_t
i = 0; i < films.size(); i++)
                        {
                            {
                                if
(films[i].id == j)
                                {

out.put("Фильм который вы хотите
изменить:");

films[i].show();

for (;;)
```

```
{

cinx(x, "Выберите свойства для его
замены:", 6, "Название фильма",

"Название студии", "Продюсеры",
"Операторы", "Жанры", "Актеры");

if (x == 0)

break;

if (x == 1)

{

out.put("Введите название:");

films[i].name.clear();

push_line(films[i].name);

}

if (x == 2)

{

out.put("Введите название
киностудии");

films[i].studio.clear();

push_line(films[i].studio);

}

if (x == 3)

{

out.put("Введите продюсеров через
Enter (-1 для прекращения ввода)");

films[i].producers.clear();

push_vector(films[i].producers);

}

if (x == 4)

{

out.put("Введите операторов через
Enter (-1 для прекращения ввода)");

films[i].opers.clear();

push_vector(films[i].opers);

}

if (x == 5)

{

out.put("Введите жанры ерез Enter (-
1 для прекращения ввода)");

films[i].genres.clear();

push_vector(films[i].genres);

}

if (x == 6)

{

out.put("Введите актеров ерез Enter
(-1 для прекращения ввода)");

films[i].actors.clear();

push_vector(films[i].actors);

}

out.put("Успех");

}

out.put("Измененный фильм:");

films[i].show();

cc();

out.put("Введите для продолжения");

getchar();

cls();

break;
                                    }
                                }
                            }
                        }
                        default:
                            break;
                        }

                        break;
                    }
                    default:
```

```
                {

                    break;
                }
                }
                if (!x)
                    break;
            }
            break;
        }
        case 2:
        {
            for (;;)
            {
                cinx(x, "Выберите
действие:", 4, "Отобразить текущие
фильмы", "Отобразить сведения фильма
по ключу",
                     "Поиск фильма",
"Редактирование фильмов");
                switch (x)
                {
                case 1:
                {
                    out.put("Список
текущих фильмов:");
                    for (Film &a :
films)
                        a.show();

                    out.put("Введите
для продолжения");
                    getchar();
                    cls();
                    break;
                }
                case 2:
                {
                    int j;
                    out.put("Введите
ключ нужного вам фильма или -1 для
выхода");
                    r_cin(j);
                    if (j == -1)
                    {
                        cls();
                        break;
                    }
                    for (Film &a :
films)
                    {
                        if (j ==
a.id)
                        {

a.show();

out.put("Введите для продолжения:");
                            cc();
getchar();
                            cls();
                            j = -1;
                        }
                    }

                    if (j != -1)
                    {

out.put("Фильма с данным ключом не
существует");

out.put("Введите для продолжения:");
                        cc();
                        getchar();
                        cls();
                    }
                    break;
                }
                case 3:
                {
                    int j;
                    string b;
                    Film a(0);

                    cinx(j,
"Выберите свойство, которое вы
введете, и по которому будет
происходить поиск:",
                         7, "ID",
                         "Название
фильма", "Название студии",
"Продюсеры", "Операторы", "Жанры",
"Актеры");
                    if (j == 0)
                    {
                        cls();
                        break;
                    }

                    if (j == 1)
                    {

out.put("Введите ID:");
                        cin >> a.id;
                    }

                    if (j == 2)
                    {

out.put("Введите Название фильма:");
                        getline(cin,
a.name);
                    }

                    if (j == 3)
                    {
```

```
out.put("Введите Название студии:");
                        getline(cin,
a.studio);
                    }

                    if (j == 4)
                    {

out.put("Введите продюсеров через
запятую:");
                        getline(cin,
b);

                        a.producers
= (e_parse(b));
                    }
                    if (j == 5)
                    {

out.put("Введите операторов через
запятую:");
                        getline(cin,
b);
                        a.opers =
(e_parse(b));
                    }
                    if (j == 6)
                    {

out.put("Введите жанры через
запятую:");
                        getline(cin,
b);
                        a.genres =
(e_parse(b));
                    }
                    if (j == 7)
                    {

out.put("Введите актеров через
запятую:");
                        getline(cin,
b);
                        a.actors =
(e_parse(b));
                    }
                    cout << endl;
                    a.find();

                    break;
                }

                case 4:
                {
                    out.put("Нажата
4");
                    cinx(x,
"Выберите действие:", 3, "Добавить
фильм", "Удалить фильм", "Изменить
фильм");

                    switch (x)
                    {
                    case 1:
                    {


films.push_back(Film());

out.put("Успешно добавлен");

out.put("Введите для продолжения:");
                        getchar();
                        cls();
                        break;
                    }
                    case 2:
                    {
                        for (;;)
                        {
                            int j;
                            int x =
rand(10);
                            cinx(j,
"Введите ключ фильма, который вы
хотите удалить или -1 для выхода:");
                            if (j ==
-1)

cls();
                            break;

                            for
(auto it = films.begin(); it <
films.end(); it++)
                            {
                                if
((*it).id == j)
                                {


out.put("Фильм который вы хотите
удалить:");

(*it).show();

cout << "Для подтверждения удаления
введите " << x << " :";

cinx(j, "");

if (j == x)

{

films.erase(it);
```

```
out.put("Успех");

j = -1337;

}

else

{

out.put("Отмена");

j = -1337;

}

break;
                                    }
                                }
                                if (j !=
-1337)
                                {

out.put("Фильм не был найден");
                                }
                            }

                            break;
                        }
                        case 3:
                        {
                            int j;
                            int x;
                            cinx(j,
"Введите ID фильма, который нужно
изменить:");
                            for (size_t
i = 0; i < films.size(); i++)
                            {
                                {
                                    if
(films[i].id == j)
                                    {

out.put("Фильм который вы хотите
изменить:");

films[i].show();

for (;;)

{

cinx(x, "Выберите свойства для его
замены:", 6, "Название фильма",

"Название студии", "Продюсеры",
"Операторы", "Жанры", "Актеры");

if (x == 0)

break;

if (x == 1)

{

out.put("Введите название:");

films[i].name.clear();

push_line(films[i].name);

}

if (x == 2)

{

out.put("Введите название
киностудии");

films[i].studio.clear();

push_line(films[i].studio);

}

if (x == 3)

{

out.put("Введите продюсеров через
Enter (-1 для прекращения ввода)");

films[i].producers.clear();

push_vector(films[i].producers);

}

if (x == 4)

{

out.put("Введите операторов через
Enter (-1 для прекращения ввода)");

films[i].opers.clear();

push_vector(films[i].opers);

}

if (x == 5)

{
```

```
out.put("Введите жанры ерез Enter (-
1 для прекращения ввода)");

films[i].genres.clear();

push_vector(films[i].genres);

}

if (x == 6)

{

out.put("Введите актеров ерез Enter
(-1 для прекращения ввода)");

films[i].actors.clear();

push_vector(films[i].actors);

}

out.put("Успех");

}

out.put("Измененный фильм:");

films[i].show();

cc();

out.put("Введите для продолжения");

getchar();

cls();

break;
                                    }
                                }
                            }
                        }
                        default:
                            break;
                        }

                        break;
                    }
                    default:
                    {
                        break;
                    }
                    }
                    if (!x)
                        break;
                }
                break;
            }
            case 3:
                for (;;)
                {
                    cinx(x, "Выберите
действие:", 4, "Отобразить текущие
репертуары", "Отобразить сведения
репертуара по ключу",
                         "Поиск
репертуара", "Редактирование
репертуаров");
                    switch (x)
                    {
                    case 1:
                    {
                        out.put("Список
текущих репертуаров:");
                        for (Repertoire
&a : repertoires)
                            a.show();
                        //cout << "ID :
" << a.id << ". Название : " <<
a.name << ". Жанр : " <<
unvector(a.genres) << "." << endl;
                        out.put("Введите
для продолжения");
                        getchar();
                        cls();
                        break;
                    }
                    case 2:
                    {
                        int j;
                        out.put("Введите
ключ нужного вам репертуара или -1
для выхода");
                        r_cin(j);
                        if (j == -1)
                        {
                            cls();
                            break;
                        }
                        for (Repertoire
&a : repertoires)
                        {
                            if (j ==
a.id)
                            {

a.show();

out.put("Введите для продолжения:");
                                cc();

getchar();
                                cls();
                                j = -1;
                            }
                        }
```

```
                    if (j != -1)
                    {

out.put("Репертуара с данным ключом
не существует");

out.put("Введите для продолжения:");
                        cc();
                        getchar();
                        cls();
                    }
                    break;
                }
                case 3:
                {
                    int j, i;
                    string b;
                    Repertoire a(0);

                    cinx(j,
"Выберите свойство, которое вы
введете, и по которому будет
происходить поиск:",
                         5, "ID",
                         "ID
фильма", "Дата", "Цена", "Свободные
места");
                    if (j == 0)
                    {
                        cls();
                        break;
                    }

                    if (j == 1)
                    {

out.put("Введите ID:");
                        cin >> a.id;
                    }

                    if (j == 2)
                    {


out.put("Введите ID фильма через
Enter или -1:");
                        while (i !=
-1)
                        {

r_cin(i);
                            a.film =
a.film->search(i);
                        }
                    }

                    if (j == 3)
                    {

out.put("Введите дату в формате
ДДММГГ (например, 30052021):");
                        getline(cin,
a.date);
                    }

                    if (j == 4)
                    {

out.put("Введите цену");
                        r_cin(i);

                        a.price = i;
                    }
                    if (j == 5)
                    {

out.put("Введите количество
свободных мест:");
                        r_cin(i);


a.free_places = i;
                    }
                    cout << endl;
                    a.find();

                    break;
                }

                case 4:
                {
                    out.put("Нажата
4");
                    cinx(x,
"Выберите действие:", 3, "Добавить
реперутар", "Удалить реперутар",
"Изменить реперутар");

                    switch (x)
                    {
                    case 1:
                    {


repertoires.push_back(Repertoire());

//films[films.size() - 1].save(); //

out.put("Успешно добавлен");

out.put("Введите для продолжения:");
                        getchar();
                        cls();
                        break;
                    }
                    case 2:
                    {
```

```
                        for (;;)
                        {
                            int j;
                            int x =
rand(10);
                            cinx(j,
"Введите ключ репертуара, который вы
хотите удалить или -1 для выхода:");
                            if (j ==
-1)
                            {

cls();

break;
                            }

                            for
(auto it = repertoires.begin(); it <
repertoires.end(); it++)
                            {
                                if
((*it).id == j)
                                {


out.put("Репертуар который вы хотите
удалить:");

(*it).show();

cout << "Для подтверждения удаления
введите " << x << " :";

cinx(j, "");

if (j == x)

{

repertoires.erase(it);

out.put("Успех");

j = -1337;

}

else

{

out.put("Отмена");

j = -1337;

}

break;
                                }
                            }
                            if (j !=
-1337)
                            {

out.put("Репертуар не был найден");
                            }
                        }

                        break;
                    }
                    case 3:
                    {
                        int j;
                        int x;
                        cinx(j,
"Введите ID репертуара, который
нужно изменить:");
                        for (size_t
i = 0; i < repertoires.size(); i++)
                        {
                            {
                                if
(repertoires[i].id == j)
                                {

out.put("Репертуар который вы хотите
изменить:");

repertoires[i].show();

for (;;)

{

cinx(x, "Выберите свойства для его
замены:", 4, "Дата",

"ID фильма", "Цена", "Кол-во
свободных мест");

if (x == 0)

break;

if (x == 1)

{

out.put("Введите дату в формате
ДДММГГ (например, 30052021):");

repertoires[i].date.clear();

push_line(repertoires[i].date);

}
```

```
if (x == 2)

{

out.put("Введите ID фильма");

r_cin(j);

repertoires[i].film =
repertoires[i].film->search(j);

}

if (x == 3)

{

out.put("Введите цену");

r_cin(repertoires[i].price);

}

if (x == 4)

{

out.put("Введите кол-во свободных
мест");

r_cin(repertoires[i].free_places);

}


out.put("Успех");

}

out.put("Измененный репертуар:");

films[i].show();

cc();

out.put("Введите для продолжения");

getchar();

cls();

break;
                                    }
                                }
                            }
                        }
                        default:
                            break;
                        }

                        break;
                    }
                    default:
                    {
                        // cout << -
12356889;
                        break;
                    }
                    }
                    if (!x)
                        break;
                }
                break;
            default:
                set_all();
                return 0;
            }
        }
    }
```

Г.2 Файл Cinema.h

```
#pragma once# ifndef Cinema_h
#define Cinema_h
#include "funcs.h"
#include <iostream>
#include <regex>
using namespace std;
class Cinema
{
        public:
                int id, places,
halls;
        string name, address,
category;
        bool state;
        vector<Repertoire*> reps;

        Cinema(Cinema
                const &a);
        Cinema(int a);
        Cinema();
        Cinema(string str);
        Cinema(int id, int places,
int halls, string name,
                string address,
string category, bool state,
vector<Repertoire*> reps);
        ~Cinema();
        void set_id();
        void input();
        void find();
        void show();
```

```
        void save();
        bool operator < (const
Cinema &a) const;
```

Г.3 Файл Cinema.cpp

```
#include "Cinema.h"
#include <iostream>
#include <vector>
#include <stdarg.h>
#include <fstream>
#include <string>
#include <regex>
#include <algorithm>
#include <iterator>
#include <ctime>
#include <random>
#include "funcs.h"
#include "Repertoire.h"

extern vector<Film> films;
extern vector<Cinema> cinemas;
extern vector<Repertoire>
repertoires;

Cinema::Cinema(Cinema const &a) :
Cinema(a.id, a.places, a.halls,
a.name, a.address, a.category,
a.state, a.reps) {}
Cinema::Cinema(int a) {}
Cinema::Cinema() { input(); }
Cinema::Cinema(string str)
{
    regex reg("<.*?>");
    vector<string> a;
    vector<string> h;
    vector<Repertoire *> c;
    a = parsing(str, reg);
    int f;
    reg = ("\\d+");
    h = parsing(reg, a[7]);
    Repertoire x(0);
    for (string &z : h)
    {
        f = stoi(z);
        if (x.search(f) != NULL)
        {

c.push_back(x.search(f));
        }
    }
    *this = Cinema(stoi(a[0]),
stoi(a[2]), stoi(a[3]), a[1], a[4],
a[5], (a[6] == "1"), c);
}
Cinema::Cinema(int id, int places,
int halls, string name,
               string address,
string category, bool state,
vector<Repertoire *> reps) : id(id),
places(places), halls(halls),
name(name),
address(address),
category(category), state(state),
reps(reps)
{
}
Cinema::~Cinema() {}

void Cinema::set_id()
{
    id = 1;
    bool z = false;
    for (;;)
    {
        z = false;
        for (Cinema const &a :
cinemas)
            if (id == a.id)
            {
                z = true;
                id++;
                continue;
            }
        if (!z)
            return;
    }
}
void Cinema::find()
    {
        vector<Cinema> _cinemas;
        if (id)
        {
            for (Cinema a : cinemas)
            {
                if (id == a.id)
                {
                    if
(!binary_search(_cinemas.begin(),
_cinemas.end(), a))
                    {
                        a.show();
                    }
                    else

_cinemas.push_back(a);
                }
            }
            cc();
        }
        if (places)
        {
            for (Cinema a : cinemas)
            {
                if (places ==
a.places)
                {
};#endif
```

```
                    if
(!binary_search(_cinemas.begin(),
_cinemas.end(), a))
                    {
                        a.show();
                    }
                    else

_cinemas.push_back(a);
                }
            }
        }
        if (halls)
        {
            for (Cinema a : cinemas)
            {
                if (halls ==
a.halls)
                {
                    if
(!binary_search(_cinemas.begin(),
_cinemas.end(), a))
                    {
                        a.show();
                    }
                    else

_cinemas.push_back(a);
                }
            }
        }
        if (state)
        {
            for (Cinema a : cinemas)
            {
                if (state ==
a.state)
                {
                    if
(!binary_search(_cinemas.begin(),
_cinemas.end(), a))
                    {
                        a.show();
                    }
                    else

_cinemas.push_back(a);
                }
            }
        }
        if (!name.empty())
        {
            for (Cinema a : cinemas)
            {
                if (name == a.name)
                {
                    if
(!binary_search(_cinemas.begin(),
_cinemas.end(), a))
                    {
                        a.show();
                    }
                    else

_cinemas.push_back(a);
                }
            }
        }
        if (!address.empty())
        {
            for (Cinema a : cinemas)
            {
                if (address ==
a.address)
                {
                    if
(!binary_search(_cinemas.begin(),
_cinemas.end(), a))
                    {
                        a.show();
                    }
                    else

_cinemas.push_back(a);
                }
            }
        }
        if (!category.empty())
        {
            for (Cinema a : cinemas)
            {
                if (category ==
a.category)
                {
                    if
(!binary_search(_cinemas.begin(),
_cinemas.end(), a))
                    {
                        a.show();
                    }
                    else

_cinemas.push_back(a);
                }
            }
        }
        if (!reps.empty())
        {
            for (Cinema a : cinemas)
            {
                if (reps == a.reps)
                {
                    if
(!binary_search(_cinemas.begin(),
_cinemas.end(), a))
                    {
                        a.show();
                    }
                    else

_cinemas.push_back(a);
                }
            }
        }
        if (_cinemas.empty())
        {
```

```
            print("Ничего не
найдено");
        }

        print("Введите для
продолжения:");
        _cinemas.clear();
        getchar();
        cls();
    }
void Cinema::show()
{
    bool flag;
    cout << "ID : " << id << ".
Название : " << name << ". Адрес : "
<< address
         << ". Категория : " <<
category << ".\n Кол-во мест : " <<
places
         << ". Кол-во залов : " <<
halls
         << ". Состояние : " << ((1
== state) ? ("Работает") : ("Не
Работает")) << ". \n\t Репертуары
:\n\n";
    for (Repertoire *&a : reps)
    {
        flag = false;
        for (Repertoire &b :
repertoires)
        {
            if (a == &b)
            {
                b.show();
                flag = true;
                break;
            }
        }

print("_____________________________
_____________________");
        if (!flag)
            print("Репертуара с
таким ID не существует");
    }
}
void Cinema::save()
{
    ofstream out("data/cinemas.db",
ios::app);
    if (out.is_open())
    {
        out << "<" << id << "><" <<
name << "><" << places << "><" <<
halls << "><" << address
            << "><" << category <<
"><" << state << "><";
        for (Repertoire *&a : reps)
        {
            out << a->id << " ";
        }
        out << ">" << endl;
    }
    out.close();
    print("Кинотеатр сохранен");
}
void Cinema::input()
{
    int i;
    vector<int> str;
    set_id();
    print("Введите название
кинотеатра:");
    push_line(name);
    print("Введите адрес
кинотеатра:");
    push_line(address);
    print("Введите категорию
кинотеатра:");
    push_line(category);
    print("Введите кол-во мест:");
    r_cin(places);
    print("Введите кол-во залов:");
    r_cin(halls);
    print("Введите 1, если кинотеатр
работает, 0 если нет:");
    r_cin(i);
    state = (i == 1);
    print("Введите ID репертуаров
через Enter или -1 для выхода:");
    push_vector(reps);
    print("Успех");
}

bool Cinema::operator<(const Cinema
&a) const

    {
        if ((this->id == a.id) &&
(this->name == a.name) && (this-
>halls == a.halls) &&
            (this->places ==
a.places) && (this->state ==
a.state) &&
            (this->address ==
a.address) && (this->category ==
a.category) && (this->reps ==
a.reps))
            return false;
        return true;
    }
```

Г.4 Файл Repertoire.h

```
#pragma once
#ifndef Repertoire_h
#define Repertoire_h

#include <iostream>
#include <vector>
#include "Film.h"
```

```
using namespace std;


class Repertoire
{
public:
    int id, price, free_places;
    string date;
    Film *film;
    Repertoire(int id, int price,
int free_places,
               string date, Film
*film);
    Repertoire(Repertoire const &a);
    Repertoire(int a);
    Repertoire();
    Repertoire(string str);
    ~Repertoire();
    Repertoire *search(int &j);
    void set_id();
    void input();
    void find();
    void save();
    void show();
    bool    operator<(const
Repertoire &a) const;
};
#endif
```

Г.5 Файл Repertoire.cpp

```
#include "Repertoire.h"
#include "funcs.h"
#include <fstream>
#define d(a) a[0] << a[1] << "." <<
a[2] << a[3] << "." << a.substr(4,
4)

class Cinema;
extern vector<Film> films;
extern vector<Cinema> cinemas;
extern            vector<Repertoire>
repertoires;

Repertoire::Repertoire(int a) {}
Repertoire::Repertoire()  {  input();
}
Repertoire::Repertoire(int  id,  int
price, int free_places,
                       string  date,
Film *film) : id(id), price(price),

free_places(free_places),
date(date), film(film)
{
}
Repertoire::Repertoire(string str)
{
    regex reg("\\w+");
    vector<string> a;
    a = parsing(reg, str);
    int d = stoi(a[1]);
    *this  =  Repertoire(stoi(a[0]),
stoi(a[2]), stoi(a[3]), a[4], film-
>search(d));
}
Repertoire::~Repertoire()
{
}
Repertoire::Repertoire(Repertoire
const    &a)    :    Repertoire(a.id,
a.price,    a.free_places,    a.date,
a.film) {}

Repertoire  *  Repertoire::search(int
&j)
{
    for     (Repertoire     &a     :
repertoires)
    {
        if (j == a.id)
        {
            return (&a);
            a.show();
        }
    }
    return 0;
}
void Repertoire::set_id()
{
    id = 1;
    bool z = false;
    for (;;)
    {
        z = false;
        for  (Repertoire  const  &a  :
repertoires)
            if (id == a.id)
            {
                z = true;
                id++;
                continue;
            }
        if (!z)
            return;
    }
}
void Repertoire::input()
{
    int i;
    set_id();
    print("Введите  дату  в  формате
ДДММГГ (например, 30052021):");
    push_line(date);
    print("Введите ID фильма");
    r_cin(i);
    film = film->search(i);
    print("Введите цену билета");
    r_cin(price);
    print("Введите  кол-во  свободных
мест");
```

```
    r_cin(free_places);
    print("Успех");
    cc;
    print("Введите                для
продолжения:");
    getchar();
    cls();
}
void Repertoire::find()
{
    vector<Repertoire> _reps;
    if (id)
    {
        for    (Repertoire    a    :
repertoires)
        {
            if (id == a.id)
            {
                if
(!binary_search(_reps.begin(),
_reps.end(), a))
                {
                    a.show();
                }
                else

_reps.push_back(a);
            }
        }
        cc();
    }
    if (price)
    {
        for    (Repertoire    a    :
repertoires)
        {
            if (price == a.price)
            {
                if
(!binary_search(_reps.begin(),
_reps.end(), a))
                {
                    a.show();
                }
                else

_reps.push_back(a);
            }
        }
        cc();
    }
    if (free_places)
    {
        for    (Repertoire    a    :
repertoires)
        {
            if    (free_places    ==
a.free_places)
            {
                if
(!binary_search(_reps.begin(),
_reps.end(), a))
                {
                    a.show();
                }
                else

_reps.push_back(a);
            }
        }
        cc();
    }
    if (!date.empty())
    {
        for    (Repertoire    a    :
repertoires)
        {
            if (date == a.date)
            {
                if
(!binary_search(_reps.begin(),
_reps.end(), a))
                {
                    a.show();
                }
                else

_reps.push_back(a);
            }
        }
        cc();
    }
    if (!date.empty())
    {
        for    (Repertoire    a    :
repertoires)
        {
            if (film == a.film)
            {
                if
(!binary_search(_reps.begin(),
_reps.end(), a))
                {
                    a.show();
                }
                else

_reps.push_back(a);
            }
        }
        cc();
    }
    _reps.clear();
    print("Введите                для
продолжения:");
    getchar();
    cls();
}
void Repertoire::save()
{
    ofstream
out("data/repertoires.db",
ios::app);
    if (out.is_open())
    {
```

```
        out << id << " " << film->id
<< " " << price << " "
             << free_places << " " <<
date << " ";
    }
    out.close();
    print("Репертуар сохранен");
}
void Repertoire::show()
{
    cout << "ID : " << id << ". Дата
: " << d(date) << ". Цена : " <<
price
         << ".\n Кол-во свободных
мест: : " << free_places << ". Фильм
: " << endl;
    for (Film &a : films)
    {
        if (film == &a)
        {
            film->show();
            return;
        }
    }
    print("Фильма с таким ID не
существует");
}

bool     Repertoire::operator<(const
Repertoire &a) const

    {
        if ((this->id == a.id) &&
(this->date == a.date) && (this-
>film == a.film) &&
            (this->free_places    ==
a.free_places) && (this->price ==
a.price))
            return false;
        return true;
    }
```

Г.6 Файл Film.h

```
#pragma once
#ifndef Film_h
#define Film_h
#include <iostream>
#include <vector>
using namespace std;
class Film
{
public:
    int id;
    string name, studio;
    vector<string> producers;
    vector<string> opers;
    vector<string> genres;
    vector<string> actors;
    Film(string str);
    Film();
    Film(int a);
    Film(Film const &a);
    Film(int id, string name, string
studio,
         vector<string> producers,
vector<string> opers,
         vector<string> genres,
vector<string> actors);
    ~Film();

    Film *search(int &j);
    void set_id();
    void input();
    void find();
    void show();
    void save();
    bool operator<(const Film &a)
const;
};

#endif
```

Г.7 Файл Film.cpp

```
#include "Film.h"
#include <iostream>
#include <vector>
#include <fstream>
#include <stdarg.h>
#include <string>
#include <regex>
#include <algorithm>
#include <iterator>
#include <ctime>
#include <random>
#include "Repertoire.h"
class Cinema;
void cc();
void print(string text);
void r_cin(int &j);
void to_lower(char *c);
string low(string s);
vector<string> low(vector<string>
x);
void reverseStr(string &str);
vector<string> e_parse(string str);
void get_films(vector<Film> &films);
void get_all();
string unvector(const vector<string>
&a);
void cls();
void menu(int i, ...);
```

```
void cc();
void cinx(int &x, string text, int
i, ...);
void cinx(int &x, string text);
void cinx(string &x, string text,
int i, ...);
void push_line(string &str);
string str(vector<string> &b);
string str(string &b);
void push_vector(vector<string> &a);
vector<string> parsing(string &str,
regex &reg);
vector<string> parsing(regex &reg,
string &str);
void get_films();
void get_repertoires();
void get_cinemas();
void get_all();
void set_all();
void push_vector(vector<Repertoire
*> &a);

// class Cinema;
// class Repertoire;

extern vector<Film> films;
extern vector<Cinema> cinemas;
extern vector<Repertoire>
repertoires;

Film::Film(Film const &a) :
id(a.id), name(a.name),
studio(a.studio),
producers(a.producers),

opers(a.opers), genres(a.genres),
actors(a.actors) {}
Film::Film(string str)
{
    regex reg("\\d+");
    sregex_iterator abc(str.begin(),
str.end(), reg);
    vector<string> a;
    reg = ("(<.*?>)");
    a = parsing(str, reg);
    reg = ("'.*?'");
    *this =
Film(stoi(smatch(*abc).str()), a[0],
a[1], parsing(a[2], reg),
parsing(a[3], reg),
                parsing(a[4], reg),
parsing(a[5], reg));
}
Film::Film() { input(); }
Film::Film(int a) {}
Film::Film(int id, string name,
string studio,
           vector<string> producers,
vector<string> opers,
           vector<string> genres,
vector<string> actors) : id(id),
name(name), studio(studio),
producers(producers),
opers(opers), genres(genres),
actors(actors) {}
Film::~Film() {}
Film * Film::search(int &j)
{

    for (Film &a : films)
    {
        if (j == a.id)
        {
            return &a;
            a.show();
        }
    }
    return 0;
}
void Film::set_id()
{
    id = 1;
    bool z = false;
    for (;;)
    {
        z = false;
        for (Film const &a : films)
            if (id == a.id)
            {
                z = true;
                id++;
                continue;
            }
        if (!z)
            return;
    }
}
void Film::input()
{
    set_id();
    print("Введите название:");
    push_line(name);
    print("Введите название
киностудии");
    push_line(studio);
    print("Введите продюсеров через
Enter (-1 для прекращения ввода)");
    push_vector(producers);
    print("Введите операторов через
Enter (-1 для прекращения ввода)");
    push_vector(opers);
    print("Введите жанры ерез Enter
(-1 для прекращения ввода)");
    push_vector(genres);
    print("Введите актеров ерез
Enter (-1 для прекращения ввода)");
    push_vector(actors);
    print("Успех");
}
void Film::find()
{
    vector<Film> _films;
    if (id)
    {
        for (Film a : films)
```

```
        {
            if (id == a.id)
            {
                if
(!binary_search(_films.begin(),
_films.end(), a))
                {
                    a.show();
                }
                else

_films.push_back(a);
            }
        }
        cc();
    }
    if (!name.empty())
    {
        for (Film a : films)
        {
            if (low(name) ==
low(a.name))
            {
                if
(!binary_search(_films.begin(),
_films.end(), a))
                {
                    a.show();
                }
                else

_films.push_back(a);
            }
        }
    }
    if (!studio.empty())
    {
        for (Film a : films)
        {
            if (low(studio) ==
low(a.studio))
            {
                if
(!binary_search(_films.begin(),
_films.end(), a))
                {
                    a.show();
                }
                else

_films.push_back(a);
            }
        }
    }
    if (!producers.empty())
    {
        for (Film a : films)
        {
            for (string &x :
producers)
            {
                for (string &y :
a.producers)
                {
                    // cout <<
low(x) << "==" << low(y) << endl;
                    if (low(x) ==
low(y))
                    {
                        if
(!binary_search(_films.begin(),
_films.end(), a))
                        {

a.show();

_films.push_back(a);
                        }
                    }
                }
            }
        }
    }
    if (!opers.empty())
    {
        for (Film a : films)
        {
            for (string &x : opers)
            {
                for (string &y :
a.opers)
                {
                    // cout <<
low(x) << "==" << low(y) << endl;
                    if (low(x) ==
low(y))
                    {
                        if
(!binary_search(_films.begin(),
_films.end(), a))
                        {

a.show();

_films.push_back(a);
                        }
                    }
                }
            }
        }
    }
    if (!genres.empty())
    {
        for (Film a : films)
        {
            for (string &x : genres)
            {
                for (string &y :
a.genres)
                {
                    if (low(x) ==
low(y))
                    {
                        if
(!(binary_search(_films.begin(),
_films.end(), a)))
```

```
                        {
a.show();

_films.push_back(a);
                        }
                    }
                }
            }
        }
    }
    if (!actors.empty())
    {
        for (Film a : films)
        {
            for (string &x : actors)
            {
                for (string &y :
a.actors)
                {
                    if (low(x) ==
low(y))
                    {
                        if
(!binary_search(_films.begin(),
_films.end(), a))
                        {

a.show();

_films.push_back(a);
                        }
                    }
                }
            }
        }
    }
    _films.clear();
    print("Введите для
продолжения:");
    getchar();
    cls();
}
void Film::show()
{
    cout << "ID : " << id << ".
Название : " << name << ". Жанр(ы) :
" << unvector(genres)
         << ". Студия : " << studio
<< ".\nПродюссер(ы) : " <<
unvector(producers)
         << ". Оператор(ы) : " <<
unvector(opers)
         << ".\nАктер(ы) : " <<
unvector(actors) << endl
         << endl;
}
void Film::save()
{
    ofstream out("data/films.db",
ios::app);
    if (out.is_open())
    {
        out << id << str(name) <<
str(studio) << str(producers)
            << str(opers) <<
str(genres) << str(actors) << endl;
    }
    out.close();
    print("Фильм сохранен");
}

bool Film::operator<(const Film &a)
const
    {
        if ((this->id == a.id) &&
(this->name == a.name) && (this-
>studio == a.studio) &&
            (this->producers ==
a.producers) && (this->opers ==
a.opers) &&
            (this->genres ==
a.genres) && (this->actors ==
a.actors))
            return false;
        return true;
    }
```

Г.8 Файл func.h

```
#pragma once
#ifndef funcs_h
#define funcs_h
#include <iostream>
#include "Repertoire.h"
#include <regex>
using namespace std;
void cc();
void print(string text);
void r_cin(int &j);
void to_lower(char *c);
string low(string s);
vector<string> low(vector<string>
x);
void reverseStr(string &str);
vector<string> e_parse(string str);
void get_films(vector<Film> &films);
void get_all();
string unvector(const vector<string>
&a);
void cls();
void menu(int i, ...);
void cc();
void cinx(int &x, string text, int
i, ...);
void cinx(int &x, string text);
void cinx(string &x, string text,
int i, ...);
```

```
void push_line(string &str);
string str(vector<string> &b);
string str(string &b);
void push_vector(vector<string> &a);
vector<string> parsing(string &str,
regex &reg);
vector<string> parsing(regex &reg,
string &str);
void get_films();
void get_repertoires();
void get_cinemas();
void get_all();
void set_all();
void push_vector(vector<Repertoire
*> &a);
#endif
```

Г.9 Файл func.cpp

```
#include "funcs.h"
#include "Film.h"
#include "Cinema.h"
#include "Repertoire.h"
#include <stdarg.h>
#include <fstream>
using namespace std;

extern vector<Film> films;
extern vector<Cinema> cinemas;
extern vector<Repertoire>
repertoires;

void cc();
void print(string text) { cout <<
text << endl; } //вывод текста
void r_cin(int &j)
{
    for (;;)
    {
        cin >> j;
        if (cin.fail())
            cc();
        else
            break;
        print("Неверный ввод,
попробуйте ввести число:");
    }
}
void to_lower(char *c)
{
    if (*c >= 'A' && *c <= 'Z')
    {
        *c = *c + 'a' - 'A';
    }
    if (*c >= 'A' && *c <= 'Я')
    {
        *c = *c + 'a' - 'A';
    }
}
string low(string s)
{
    for (size_t i = 0; i < s.size();
i++)
    {
        to_lower(&s[i]);
    }
    return s;
}
vector<string> low(vector<string> x)
{
    for (string &s : x)
        for (size_t i = 0; i <
s.size(); i++)
            s[i] = tolower(s[i],
locale("ru"));

    for (string &s : x)
        cout << s << endl;
    return x;
}
void reverseStr(string &str)
{
    int n = str.length();

    // Swap character starting from
two
    // corners
    for (int i = 0; i < n / 2; i++)
        swap(str[i], str[n - i -
1]);
}
vector<string> e_parse(string str)
{
    int i = 0, j, y;
    string word;
    vector<string> a;
    while (i < str.size())
    {
        word = "";
        while (str[i] != ',' && i <
str.size())
        {
            // cout << "1 " <<
str[i] << endl;
            word += str[i];
            i++;
        }
        j = 0;
        // cout << "1.5 " << word[j]
<< endl;
        if (word[j] == ' ')
        {
            for (j = 0; j <
word.size(); j++)
            {
                // cout << "2 " <<
word[j] << endl;
                if (word[j] != ' ')
                {
                    j +=
word.size();
```

```
                }
            };
            // cout << "2.5 " <<
word[j] << endl;
            word = word.substr(j -
word.size() - 1);
        }
        for (j = 0; j < word.size();
j++)
        {
            // cout << "3 " <<
word[j] << endl;
            if (word[j] == ' ')
            {
                break;
            }
        };
        for (y = j; y < word.size();
y++)
        {
            // cout << "4 " <<
word[y] << endl;
            if (word[y] != ' ')
            {
                break;
            }
        };
        word = word.substr(0, j + 1)
+ word.substr(y);

        reverseStr(word);
        j = 0;
        // cout << "3.5 " << word[j]
<< endl;
        if (word[j] == ' ')
        {
            for (j = 0; j <
word.size(); j++)
            {
                // cout << "2 " <<
word[j] << endl;
                if (word[j] != ' ')
                {
                    j +=
word.size();
                }
            };
            // cout << "2.5 " <<
word[j] << endl;
            word = word.substr(j -
word.size() - 1);
        }
        reverseStr(word);
        a.push_back(word);
        i++;
    }
    // for(string b:a)cout<<b<<endl;
    return a;
}
void get_films(vector<Film> &films);
void get_all(vector<Cinema>
&cinemas, vector<Film> &films,
vector<Repertoire> &repertoires);
```

```
string unvector(const vector<string>
&a)
{
    string s;
    for (string const &b : a)
        s += b + ((b != a[a.size() -
1]) ? ", " : "");
    return s;
}
void cls() { printf("\e[1;1H\e[2J");
} //очистка консоли
void menu(int i, ...)
//контекстное меню
{
    va_list t;
    va_start(t, i);
    print("Выберите действие:");
    for (size_t j = 0; j <= i; j++)
        printf("[%d] - %s\n", j, (j
? va_arg(t, char *) : "Вернуться
(выход)"));
}
void cc() //очистка потока ввода
{
    cin.clear();
    cin.ignore(32767, '\n');
}
void cinx(int &x, string text, int
i, ...) //ввод x
{

    print(text);
    va_list t;
    va_start(t, i);
    for (size_t j = 0; j <= i; j++)
        printf("[%d] - %s\n", j, (j
? va_arg(t, char *) : "Вернуться
(выход)"));
    cin >> x;
    while (cin.fail() || (x < 0 || x
> i))
    {
        cc();
        cout << "Неверное значение
(введите цифру от 1 до " << i << ")"
<< endl;
        cin >> x;
    }
    cc();
    cls();
}
void cinx(int &x, string text)
{
    print(text);
    cin >> x;
    while (cin.fail())
    {
        cc();
        cout << "Неверное значение
(введите число)" << endl;
        cin >> x;
    }
}
```

```
void cinx(string &x, string text,
int i, ...) //ввод x
{

    print(text);
    va_list t;
    va_start(t, i);
    for (size_t j = 0; j <= i; j++)
        printf("[%d] - %s\n", j, (j
? va_arg(t, char *) : "Вернуться
(выход)"));

}
void push_line(string &str) //ввод
не пустых строк
{
    for (;;)
    {
        getline(cin, str);
        string a;
        for (size_t i = 0; i <
str.size(); i++)
        {
            if (str[i] != ' ')
                a += str[i];
        }
        if (a == "")
            print("Err: empty
line");
        else
            return;
    }
}
string str(vector<string> &b)
//"<%s>",b[]
{
    string a = "<";
    for (string &x : b)
        a += "'" + x + "'";

    return a + ">";
}
string str(string &b) //"<%s>",b
{
    return "<" + b + ">";
}
void push_vector(vector<string> &a)
{
    string b;
    string x;
    for (;;)
    {
        // cout << "_" << b << "_"
<< endl;
        getline(cin, b);
        x = "";
        for (size_t i = 0; i <
b.size(); i++)
        {
            if (b[i] != ' ')
                x += b[i];
        }
        // cout << "_" << b << "_"
<< endl;
        if (x == "")
        {
            print("Err: empty
line");
            continue;
        }
        if (b == "-1" && a.size() >
0)
            break;
        a.push_back(b);
    }
}
vector<string> parsing(string &str,
regex &reg)
{
    string s;
    // int i == 0;
    vector<string> a;
    sregex_iterator
currentMatch(str.begin(), str.end(),
reg);
    sregex_iterator lastMatch;
    while (currentMatch !=
lastMatch)
    {
        s = "";
        smatch match =
*currentMatch;
        for (size_t i = 0; i <
match.str().size(); i++)
        {
            if (i <
match.str().size() - 2)
                s += match.str()[i +
1];
        }
        a.push_back(s);
        currentMatch++;
    }
    if (a.size() == 7)
        a.push_back("0");
    return a;
}
vector<string> parsing(regex &reg,
string &str)
{
    string s;
    vector<string> a;
    sregex_iterator
currentMatch(str.begin(), str.end(),
reg);
    sregex_iterator lastMatch;
    while (currentMatch !=
lastMatch)
    {
        smatch match =
*currentMatch;
        a.push_back(match.str());
        currentMatch++;
    }
    return a;
```

```
}
void get_films()
{
    films.clear();
    string line;
    ifstream in("data/films.db"); //
окрываем файл для чтения
    if (in.is_open())
    {
        while (getline(in, line))
        {

films.push_back(Film(line));
        }
        in.close();
    }
}
void get_repertoires()
{
    repertoires.clear();
    string line;
    ifstream
in("data/repertoires.db"); //
окрываем файл для чтения
    if (in.is_open())
    {
        while (getline(in, line))
        {

repertoires.push_back(Repertoire(lin
e));
        }
        in.close();
    }
}
void get_cinemas()
{
    cinemas.clear();
    string line;
    ifstream in("data/cinemas.db");
// окрываем файл для чтения
    if (in.is_open())
    {
        while (getline(in, line))
        {

cinemas.push_back(Cinema(line));
        }
        in.close();
    }
}
void get_all(/*vector<Cinema>
&cinemas, vector<Film> &films,
vector<Repertoire> &repertoires*/)
//выгрузка данных
{
    get_films();
    get_repertoires();
    get_cinemas();
    print("Данные успешно были
загружены");
}
void set_all(/*vector<Cinema>
&cinemas, vector<Film> &films,
vector<Repertoire> &repertoires*/)
//сохранение данных
{
    ofstream out;               //
поток для записи
    out.open("data/films.db"); //
окрываем файл для записи
    out.close();
    out.open("data/films.db");

    if (out.is_open())
    {
        for (Film &a : films)
        {
            a.save();
        }
        out.close();
    }
    out.open("data/repertoires.db");
// окрываем файл для записи
    out.close();
    out.open("data/repertoires.db");

    if (out.is_open())
    {
        for (Repertoire &a :
repertoires)
        {
            a.save();
        }
        out.close();
    }
    out.open("data/cinemas.db"); //
окрываем файл для записи
    out.close();
    out.open("data/cinemas.db");

    if (out.is_open())
    {
        for (Cinema &a : cinemas)
        {
            a.save();
        }
        out.close();
    }
    print("Соединение прооэшоло
успешно");
}
void push_vector(vector<Repertoire
*> &a)
{

    int x;
    for (;;)
    {
        r_cin(x);
        Repertoire z(0);
        // Repertoire *ptr;
        if (x != -1)
        {
            if (z.search(x) != NULL)
```

```
a.push_back(z.search(x));
        }
        else
                return;
    }
}
```

Г.10 Файл Output.h

```
#pragma once
#ifndef Output_h
#define Output_h
#include <iostream>
class Output
{
public:
    std::string out;
    Output(std::string const a);
    Output();
    ~Output();
    void put(std::string a);
    void print();
};
#endif
```

Г.11 Файл Output.cpp

```
#include "Output.h"
#include <iostream>

Output::Output(std::string a) : out(a) { print(); }
Output::Output() {}
Output::~Output() {}
void Output::put(std::string a) { std::cout << a << std::endl; }
void Output::print() { std::cout << out<<std::endl; }
```