Автономная некоммерческая организация высшего образования

«Российский новый университет»

Факультет информационных систем и компьютерных технологий

Курсовая работа

по дисциплине: Языки и методы программирования \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_

на тему: Информационная система «Фильмотека»\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_

Студента группы 424, 2 курса

Очной формы обучения

Направления подготовки:

Прикладная математика и информатика\_

Направленности (профиля):

Математическое моделирование и вычислительная математика

Вишневского Дениса Владиславовича.

(фамилия, имя, отчество)

Руководитель:

Маслянкин Вячеслав Иванович, профессор

(должность, звание, фамилия, имя, отчество)

Работа допущена к защите «\_\_\_»\_\_\_\_\_\_20\_\_г.

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Регистрационный номер\_\_\_\_\_\_\_\_ от «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_20\_\_\_г.

Москва

2019 г.

Содержание……………………………………………………………………….2

1. Введение …………………………………………………………………………………..…..3

1.1 Информационные (справочные) системы………………………………………….3

1.2 Системы (виды) программирования…………….…………………………………х

2. Теоретическая часть…………………………………………………………………………..х

2.1. Постановка задачи…………………………………………………………………..х

2.2. Описание технологии разработки системы (среда и язык программирования)..х

3. Разработка проекта …………………………………………………………………………..х

3.1. Разработка приложения………………………………………………………….…х

3.2. Разработка пользовательского интерфейса…………………………………….…х

4. Реализация проекта …………………………………………………………………………..х

5. Тестирование системы……………………………………………………………………….х

6. Заключение……………………………………………………………………………………х

Список литературы……………………………………………………………………………..х

Приложение: Листинг программы…………………………………………………………….х

**1. Введение**

**1.1 Информационные (справочные) системы**

Термин информационная система (ИС) используется как в широком, так и в узком смысле. В широком смысле информационная система есть совокупность технического, программного и организационного обеспечения, а также персонала, предназначенная для того, чтобы своевременно обеспечивать надлежащих людей надлежащей информацией.

Основное назначение информационной системы – создание современной инфраструктуры для управления предприятием, организацией, учреждением.

Задачи:

* разработка базы данных, предназначенной для хранения информации
* обработка данных об операциях, производимых фирмой;
* создание периодических контрольных отчетов о состоянии дел в фирме;
* получение ответов на всевозможные текущие запросы и оформление их в виде бумажных документов или отчетов.

Работа по созданию ИС, завершается этапом программирования, т.е. – разработкой макета программного обеспечения ИС с помощью выбранной среды. На данном этапе, скорее всего, потребуются такие действия, как:

* обоснование и выбор среды разработки;
* разработка ПО расчетно-аналитических модулей;
* разработка пользовательского интерфейса макета ИС (формы, запросы, отчеты, диалоговые окна, меню);
* формирование архитектуры и обеспечение прав доступа различных пользователей ИС.

При этом, необходимо иметь в виду, что в качестве средств для разработки ИС могут выступать:

* обычные системы программирования, основанные на языках высокого уровня (в том числе и объектно-ориентированное программирование);
* табличные процессоры (электронные таблицы);
* системы управления базами данных (СУБД);
* интегральные среды визуального программирования, основанные на языках высокого уровня;

**1.2 Виды (парадигмы) программирования**

Виды:

* Императивное программирование
* Декларативное программирование
* Структурное программирование
* Функциональное программирование
* Логическое программирование
* Объектно-ориентированное программирование:
  + 1. Компонентно-ориентированное программирование
    2. Прототипно-ориентированное программирование
    3. Агентно-ориентированное программирование

**Структурное программирование** – это метод, предполагающий создание улучшенных программ. Он служит для организации проектирования и кодирования программ таким образом, чтобы предотвратить большинство логических ошибок и обнаружить те, которые допущены.

Структурное программирование подразумевает точно обозначенные управляющие структуры, программные блоки, отсутствие инструкций GOTO, автономные подпрограммы, в которых поддерживается рекурсия и локальные переменные. Главным в структурном программировании является возможность разбиения программы на составляющие ее элементы.

**Процедурное или императивное(от лат. imperativus — повелительный) программирование** является отражением архитектуры традиционных ЭВМ. Теоретической моделью процедурного программирования служит алгоритмическая система под названием «машина Тьюринга».

Программа на процедурном языке программирования состоит из последовательности операторов (инструкций), задающих процедуру решения задачи. Основным является оператор присваивания, служащий для изменения содержимого областей памяти. Концепция памяти как хранилища значений, содержимое которого может обновляться операторами программы, является фундаментальной в императивном программировании.

Выполнение программы сводится к последовательному выполнению операторов с целью преобразования исходного состояния памяти, то есть значений исходных данных, в заключительное, то есть в результаты. Таким образом, с точки зрения программиста имеются программа и память, причем первая последовательно обновляет содержимое последней.

**Объектно-ориентированное программирование**(ООП) организует данные и алгоритмы, обрабатываемые программой. При этом программист создает формы данных и алгоритмы, соответствующие основным характеристикам решаемой проблемы. Модели данных и алгоритмы, их обрабатывающие, называются **классами**, а **объекты** — это конкретные их представители, используемые в программе.

Из общих объектов создаются другие, более специализированные. Механизм создания таких подобъектов называется **наследованием**. В итоге данные программы представляют из себя **объектную модель** — дерево объектов, начиная с самого верхнего наиболее абстрактного и общего объекта.

ООП сочетает лучшие принципы структурного программирования с новыми мощными концепциями, базовые из которых называются инкапсуляцией, полиморфизмом и наследованием.

ООП позволяет оптимально организовывать программы, разбивая проблему на составные части, и работая с каждой по отдельности.

**Объектно-ориентированное программирование** — это развитие технологии структурного программирования, однако оно имеет свои характерные черты. Основной единицей в объектно-ориентированном программировании выступает объект, который заключает в себе, инкапсулирует как описывающие его данные (свойства), так и средства обработки этих данных (методы). В системах ООП обычно используется графический интерфейс, который позволяет визуализировать процесс программирования. Появляется возможность создавать объекты, задавать им свойства и поведение с помощью мыши.

**Объект** – это комбинация данных и кода. Другими словами, объект, называемый ещё **представителем** (какого-нибудь класса), - это порция данных, значение которых определяют его текущее состояние, и набор подпрограмм, называемых **методами**, оперирующих с этими данными и определяющими **поведение** объекта, т.е. его реакцию на внешние воздействия.

**Объект состоит из следующих трех частей:**

- имя объекта;

- состояние (переменные состояния);

- методы (операции).

**Процедурный язык программирования** предоставляет возможность программисту определять каждый шаг в процессе решения задачи. Особенность таких языков программирования состоит в том, что задачи разбиваются на шаги и решаются шаг за шагом. Используя процедурный язык, программист определяет языковые конструкции для выполнения последовательности алгоритмических шагов.

Программа на процедурном языке программирования состоит из последовательности операторов(инструкций), задающих процедуру решения задачи. Выполнение программы сводится к последовательному выполнению операторов с целью преобразования исходного состояния памяти, то есть значений исходных данных, в заключительное, то есть в результаты.

Пример процедурных языков программирования: Паскаль, Бейсик, C++, C# и т.д.

**2. Теоретическая часть**

**2.1. Постановка задачи**

Информационно-справочная «Фильмотека» предназначена для регистрации и учета фильмов. Программа может быть использована частными лицами, а так же в заинтересованных организациях и учреждениях.

Предлагается учитывать такие характеристики как:

- Название фильма – например, Терминатор, Золушка, Форсаж. Данное поле представлено текстовым значением.

- Длительность. Численное значение, вводится в минутах — 120, 60 и т.д.

- Год. Например: 1998, 2005 и т. п. Численное значение, вводится в годах.

- Жанр. Текстовое значение – боевик, фильм-сказка и т. п.

- Ведущий актер (звезда). Например: Арнольд\_Шварцнеггер, Эмма\_Стоун, Вин\_Дизель . Также может быть представлено как текстовое значение.

В разрабатываемой информационно-справочной системе предусматривается возможность сохранения данных и их загрузки из файлов; добавление, удаление и редактирование записей, а также сортировки и поиска событий по ряду критериев.

**2.2. Описание технологии разработки системы**

Была выбрана интегрированная среда разработки Embarcadero (Borland) C++ Builder и язык программирования С.

Язык "C"(произносится "си").

Си - это язык программирования, широко распространенный в настоящее время среди системных программистов и разработчиков прикладных программ.

Си обеспечивает полный набор операторов структурного программирования и предлагает необычно большой набор операций. Многие операции Си соответствуют машинным командам, и поэтому допускают прямую трансляцию в машинный код. Разнообразие операций позволяет выбирать их различные наборы для минимизации результирующего кода.

Си поддерживает указатели на переменные и функции. Указатель на объект программы соответствует машинному адресу этого объекта.

Си поддерживает арифметику указателей, и тем самым позволяет осуществлять непосредственный доступ и манипуляции с адресами памяти.

**Функции***.* Программа на Си - это набор функций (процедура в Си является частным случаем функции).

Главная функция программы должна иметь имя main (главная), и именно с нее начинается выполнение программы. В программе на Си должна быть только одна функция с именем main.

Упрощенная общая форма записи функции: <имя функции>()

{

<внутренние описания>; <действия>;

}

Круглые скобки после имени обязательны. Они показывают, что это имя функции, а не имя переменной.

Тело функции размещается в фигурных скобках и включает в себя описания переменных и выполняемые над ними действия.

**Описание переменных**. Каждая переменная, которая используется в программе на Си, должна быть описана ровно один раз. Описание начинается с указания типа, затем через запятую перечисляются имена переменных этого типа. В конце ставится точка с запятой.

**Определение констант***.* Определение констант обеспечивается препроцессором языка Си и имеет вид:

#define <имя константы> <значение константы>

#define <имя константы> <имя константы>

#define <имя константы> (<выражение из констант>)

Знак "#" записывается в первой позиции строки.

**Операторы***.* В Си различают простые, составные операторы и пустой оператор.

Пример простого оператора – это оператор присваивания. В языке Си он имеет вид:

имя\_переменной = выражение

Выполнение оператора присваивания: вычисляется значение выражения, указанного справа от знака «=», и это значение присваивается переменной, имя которой указано слева от знака «=».

При записи выражения в операторе присваивания используются стандартные арифметические операции "+", "-", "\*", "/" .

Пустой оператор - это просто ";". Используется пустой оператор обычно в случае, если по синтаксису надо выполнить оператор, а по алгоритму ничего не надо делать.

Любой оператор может быть помечен меткой. Метка - это идентификатор, который записывается перед оператором и отделяется от него двоеточием. В пределах одной программы метки должны быть уникальны.

**Точка с запятой***.* В языке Си точка с запятой ";" обозначает конец описания или оператора. Таким образом, каждый отдельный оператор должен заканчиваться точкой с запятой. Исключением является составной оператор: набор операторов, заключенных в фигурные скобки. После составного оператор точка с запятой не ставится.

**Комментарии**. Комментарии в Си начинаются парой знаков "/\*", оканчиваются парой знаков "\*/". Один комментарий может занимать несколько строк. Вложенность комментариев в стандартном Си не допускалась, но в большинстве современных компиляторов возможна.

В языке С++ существует еще один вид комментариев - однострочный комментарий. Все символы, расположенные за парой знаков "//" и до конца строки, рассматриваются как комментарий.

**Препроцессор языка Си**. Препроцессор используется для обработки текста программы до этапа ее компиляции. Как правило, этап препроцессорной обработки и компиляции объединяют в одну программу и называют ее просто компилятором.

Инструкции препроцессора начинаются со знака "#" в первой позиции строки. Инструкции препроцессора могут появляться в любом месте программы, и их действие распространяется до конца файла, если оно не прекращено другими инструкциями. Препроцессор обеспечивает:

1)определение текста с помощью макроподстановок - команда #define;

2)включение в текст файлов - команда #include.

**Символьные константы***.* В персональном компьютере символы хранятся в виде кодов. Соответствие между каждым символом и его кодом задается специальной кодовой таблицей. На нее разработан стандарт ASCII, поэтому коды символов называют ASCII-кодами.

Для представления каждого символа в персональном компьютере используется один байт, поэтому общее число символов равно 28 = 256. Кодовая таблица, которая устанавливает соответствие между символом и его кодом, имеет 256 строк вида:

код\_символа\_в\_заданной\_системе\_счисления - символ.

Важным понятием языка является идентификатор, который используется в качестве имени объекта (функции, переменной, константы и др.). Идентификаторы должны выбираться с учетом следующих правил:

* Они должны начинаться с буквы латинского алфавита (а,...,z, А,...,Z) или с символа подчеркивания (\_).
* В них могут использоваться буквы латинского алфавита, символ подчеркивания и цифры (0,...,9). Использование других символов в идентификаторах запрещено.
* В языке Си буквы нижнего регистра (а,...,z), применяемые в идентификаторах, отличаются от букв верхнего регистра (А,...,Z). Это означает, что следующие идентификаторы считаются разными: name, NaMe, NAME и т.д.
* Идентификаторы могут иметь любую длину, но воспринимается и используется для различения объектов (функций, переменных, констант и т.д.) только часть символов. Их число меняется для разных систем программирования, но в соответствии со стандартом ANSI C не превышает 32 (в Си++ это ограничение снято). Если длина идентификатора установлена равной 5, то имена count и counter будут идентичны, поскольку у них совпадают первые пять символов.
* Идентификаторы для новых объектов не должны совпадать с ключевыми словами языка и именами стандартных функций из библиотеки.

**Embarcadero RAD Studio** — среда быстрой разработки приложений (RAD) фирмы Embarcadero Technologies, работающая под Windows. Написана на Object Pascal и C++.

**Embarcadero C++ Builder** — это среда *C++*, которая полностью соответствует концепции быстрой разработки приложений, объединяет средства *ANSI C++* и многофункциональную расширяемую инфраструктуру визуальных компонентов.

**3. Разработка проекта**

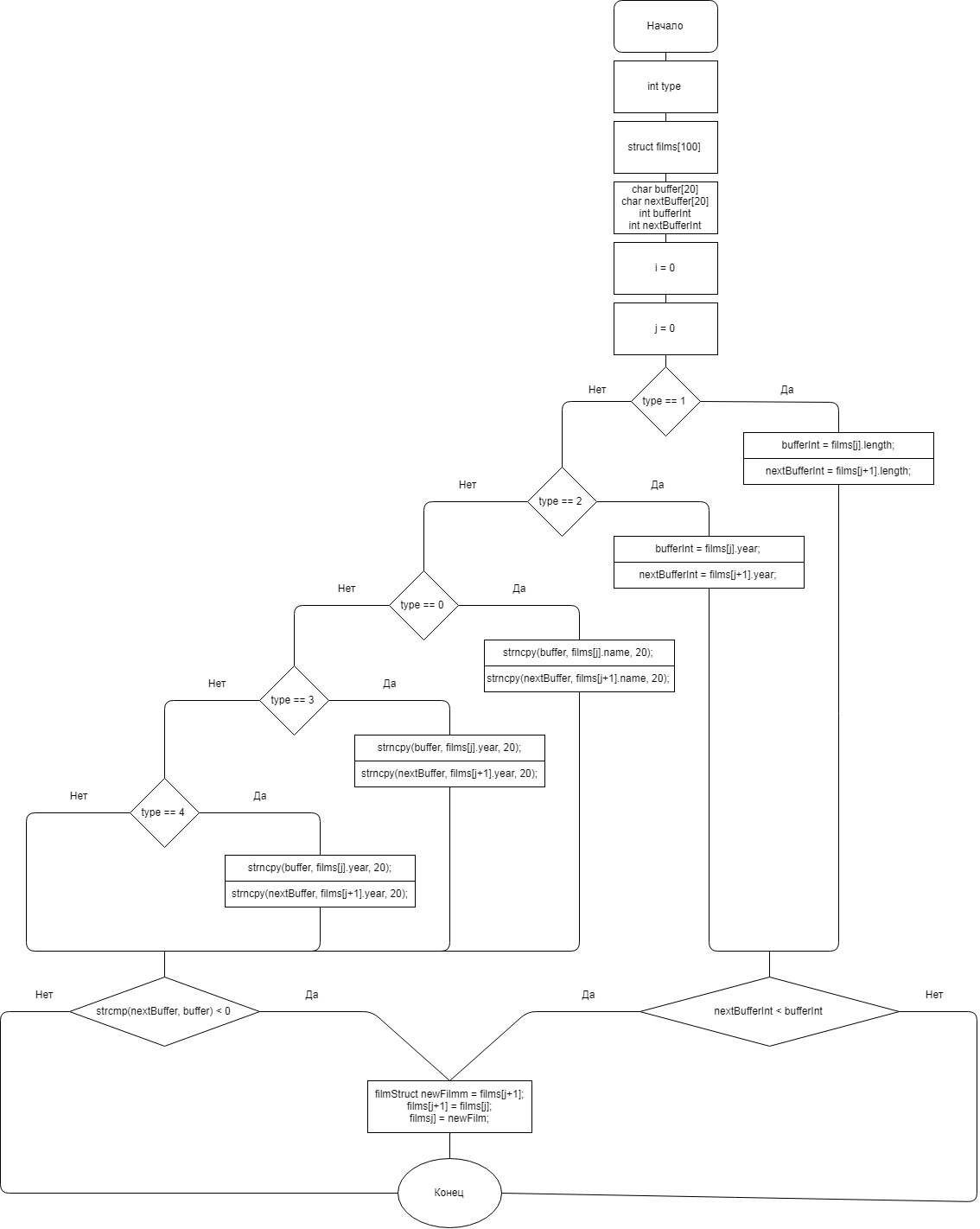
**3.1. Разработка приложения**

Структура, описывающая объект рассматриваемой предметной области имеет вид:

struct filmStruct  
{  
 char name[20];  
 int length[20];  
 int year[20];  
 char style[20];  
 char actor[20];  
};

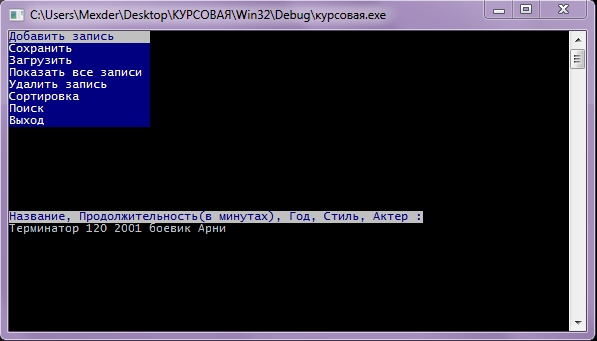
Записи по всем фильмам образуют таблицу. Соответствующая информация заносится в массив фиксированной длины.

Блок-схема последовательного поиска: сортировка «пузырек»



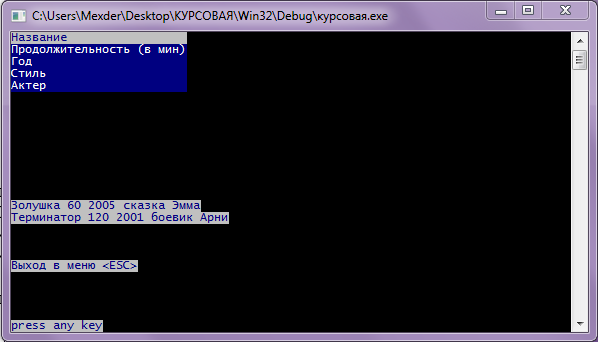
**3.2. Разработка пользовательского интерфейса**

Главное окно приложения имеет вид:



На рисунке представлено главное меню программы, поверх которого выведено сообщение, выведенное в результате выполнения команды «Open». Главное меню содержит команды «Добавить запись», «Сохранить», «Загрузить», «Показать все записи», «Удалить запись», «Сортировка», «Поиск» и «Выход». Команды меню продублированы горячими клавишами.

На втором рисунке представлено вложенное меню, вызываемое командами «Сортировка» и «Поиск» главного меню.



**4. Реализация проекта**

Данный проект выполнен в среде Embarcadero XE5 Builder C++. Проект состоит из двух основных файлов (здесь приведены имена файлов, задаваемые по умолчанию):

- курсовая.cbproj – файл проекта

- курсовая.cpp – реализация основных операций

Структура, описывающая объект рассматриваемой предметной области имеет вид:

struct filmStruct  
{  
 char name[20];  
 int length[20];  
 int year[20];  
 char style[20];  
 char actor[20];  
};

Записи по всем фильмам образуют плоскую таблицу (реляционная модель данных). Соответствующая информация заносится в массив фиксированной длины.

Данные хранятся в бинарном файле.

Упорядочивания записей осуществляется с помощью алгоритма сортировки методом «пузырька». Для поиска используется алгоритм последовательного поиска.

Соответствующие блок-схемы и коды приведены в разделе 3.1 и приложении пояснительной записки.

**5. Тестирование системы**

Было выполнено тестирование разработанного программного обеспечения с целью обнаружения возможных ошибок в ходе выполнения программы. Тестирование обеспечивает:

- обнаружение ошибок;

- демонстрацию соответствия функций программы ее назначению;

- отображение надежности функционирования программы.

Применялось функциональное тестирование, основанное на принципе «черного ящика», обеспечивающее поиск следующих категорий ошибок:

- некорректные или не реализованные функции;

- ошибки интерфейса;

- ошибки в структурах данных и в доступе к внешним данным;

- ошибки инициализации или завершения.

Тестирование проводилось в операционной системе Windows 7, с использованием возможностей интегрированной среды Embarcadero XE5 Builder C++.

Порядок испытаний - итеративный ввод граничных и некорректных значений. В частности, продолжительность фильма должна указываться в минутах, а дата выпуска в годах. Эти требования обеспечиваются верификацией вводимых пользователем данных в ходе выполнения программы.

Кроме того, проверялась корректное выполнение всех доступных пользователю операций: добавление, удаление и редактирование записей; загрузка данных из внешнего файла и сохранение информации; операций связанных с поиском и упорядочиванием записей.

**6. Заключение**

Была спроектирована и разработана информационно-справочная система «Фильмотека» предназначенная для регистрации и учета фильмов. Создано работоспособное приложение с использованием дружеского интерфейса. Предусмотрена проверка вводимых значений на корректность, протестированы операции. Программа демонстрирует принципы процедурного программирования и возможности интегрированной среды Embarcadero XE5 Builder С++.

Программа может быть использована частными лицами, а так же в заинтересованных организациях и учреждениях.

В дальнейшем возможна доработка алгоритмов сортировки, поиска и фильтрации. Возможно добавление новых характеристик в базу данных, переработка функций, реализующих основные операции, а также модернизация интерфейса программы.

**Список литературы**

1. Литература по информационным системам (информация из Интернета, в том числе [ru.wikipedia.org](https://ru.wikipedia.org/) ).

2. Литература по Builder C++ (Культин Н.Б., “Самоучитель C++ Builder”, стр. 38, 134).

3. Литература по C (Е.В.Егорова, “Программирование на языке Си”, стр. 4, 5, 12, 14, 15, 16; Х. и П. Дейтел, “Как программировать на C”, стр. 36, 38, 43)

**Приложение: Листинг программы**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**курсовая.cpp**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <stdio.h>

#include <conio.h>

#include <windows.h>

#include <math.h>

#include <stdlib.h>

#define ESC 27

#define ENTER 13

#define UP 72

#define DOWN 80

#define LEFT 75

#define RIGHT 77

int TOTAL\_FILMS\_COUNT = 0;

struct filmStruct

{

char name[20];

int length;

int year;

char style[20];

char actor[20];

};

//Ожидание отклика

void waitForRequest();

//Добавить фильм

void addFilm(filmStruct \*films);

//Сохранение и загрузка БД

void saveDataBase(filmStruct \*films);

void loadDataBase(filmStruct \*films);

//Глобальная обработка (Сортировка и поиск)

void commandTreatment(filmStruct \*films, int type);

//Обработка поиска

void searchChooseTreatment(filmStruct \*films, int type);

//Обработка сортировки

void sortChooseTreatment(filmStruct \*films, int type);

//Показать все фильмы

void showAllFilms(filmStruct \*films);

//Удаление фильмов

void deleteFilm(filmStruct \*films);

//Перерисовка окна

void repaintFrame(char item[5][100], int currentType);

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

char menu[][100] =

{

"Добавить запись ",

"Сохранить ",

"Загрузить ",

"Показать все записи ",

"Удалить запись ",

"Сортировка ",

"Поиск ",

"Выход ",

};

//Создаем структуру

filmStruct films[100];

int current = 0, code;

cprintf("Выход в меню <ESC>");

do

{

textbackground(BLACK);

clrscr();

textbackground(BLUE);

textcolor(WHITE);

for (int i = 0; i < 8; i++)

{

gotoxy(1, i + 1);

cputs(menu[i]);

}

textbackground(WHITE);

textcolor(BLUE);

gotoxy(1, current + 1);

cputs(menu[current]);

code = getch();

switch (code)

{

case UP:

if (current > 0)

current--;

else

current = 7;

break;

case DOWN:

if (current < 7)

current++;

else

current = 0;

break;

case ESC:

return 0;

case ENTER:

switch (current)

{

case 0:

addFilm(films);

break;

case 1:

saveDataBase(films);

break;

case 2:

loadDataBase(films);

break;

case 3:

showAllFilms(films);

break;

case 4:

deleteFilm(films);

break;

case 5:

commandTreatment(films, 1);

break;

case 6:

commandTreatment(films , 0);

break;

case 7:

return 0;

}

}

}while (code != 27);

return 0;

}

void waitForRequest()

{

gotoxy(1,25);

cputs("press any key");

getch();

}

void addFilm(filmStruct \*films)

{

gotoxy(1,15);

puts("");

cprintf("Название, Продолжительность(в минутах), Год, Стиль, Актер :");

gotoxy(1,17);

scanf("%s%d%d%s%s", films[TOTAL\_FILMS\_COUNT].name, &films[TOTAL\_FILMS\_COUNT].length,

&films[TOTAL\_FILMS\_COUNT].year,films[TOTAL\_FILMS\_COUNT].style, films[TOTAL\_FILMS\_COUNT].actor);

TOTAL\_FILMS\_COUNT++;

waitForRequest();

}

void saveDataBase(filmStruct \*films)

{

FILE\* file = fopen("DataBase.txt", "wb");

fwrite(films, sizeof(filmStruct), TOTAL\_FILMS\_COUNT, file);

fclose(file);

waitForRequest();

}

void loadDataBase(filmStruct \*films)

{

gotoxy(1,15);

FILE\* file = fopen("DataBase.txt", "rb");

TOTAL\_FILMS\_COUNT = fread(films, sizeof(filmStruct), 100, file);

fclose(file);

cprintf("Кол-во записей о фильмах : %i", TOTAL\_FILMS\_COUNT);

waitForRequest();

}

void showAllFilms(filmStruct \*films)

{

if (TOTAL\_FILMS\_COUNT == 0)

{

gotoxy(1,15);

cputs("Нет записей");

}

else

{

gotoxy(1,17);

for (int i=0; i < TOTAL\_FILMS\_COUNT; i++)

{

gotoxy(1, i + 15);

cprintf("%s %d %d %s %s\n", films[i].name, films[i].length, films[i].year,

films[i].style, films[i].actor);

}

}

waitForRequest();

}

void deleteFilm(filmStruct \*films)

{

gotoxy(1,17);

int current;

cprintf("Введите номер по списку : ");

scanf("%i", &current);

if (current > TOTAL\_FILMS\_COUNT || current < 0)

{

cprintf("Нет такой записи в списке");

waitForRequest();

return;

}

else

{

for(int i = current ; i < TOTAL\_FILMS\_COUNT - 1 ; i++)

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

TOTAL\_FILMS\_COUNT--;

showAllFilms(films);

waitForRequest();

}

}

void commandTreatment(filmStruct \*films, int type)

{

gotoxy(1,15);

int currentType = 0, code;

char item[5][100] =

{

"Название ",

"Продолжительность (в мин)",

"Год ",

"Стиль ",

"Актер ",

};

do

{

repaintFrame(item, currentType);

code = getch();

switch(code)

{

case UP:

if (currentType > 0)

currentType--;

else

currentType = 4;

break;

case DOWN:

if (currentType < 4)

currentType++;

else

currentType = 0;

break;

case ESC:

return;

case ENTER:

if(type == 0)

searchChooseTreatment(films, currentType);

else

sortChooseTreatment(films, currentType);

break;

}

} while(code != 27);

}

void sortChooseTreatment(filmStruct \*films, int type)

{

if(type == 1 || type == 2)

{

int bufferInt;

int nextBufferInt;

for(int i = 0 ; i < TOTAL\_FILMS\_COUNT - 1 ; i++)

for(int j = 0 ; j < TOTAL\_FILMS\_COUNT - 1 ; j++)

{

if(type == 1)

{

bufferInt = films[j].length;

nextBufferInt = films[j+1].length;

}

else

{

bufferInt = films[j].year;

nextBufferInt = films[j+1].year;

}

if(nextBufferInt < bufferInt)

{

filmStruct newFilm = films[j+1];

films[j+1] = films[j];

films[j] = newFilm;

}

}

}

else

{

char buffer[20];

char nextBuffer[20];

for(int i = 0 ; i < TOTAL\_FILMS\_COUNT - 1 ; i++)

for(int j = 0 ; j < TOTAL\_FILMS\_COUNT - 1 ; j++)

{

if(type == 0)

{

strncpy(buffer, films[j].name, 20);

strncpy(nextBuffer, films[j+1].name, 20);

}

else if(type == 3)

{

strncpy(buffer, films[j].style, 20);

strncpy(nextBuffer, films[j+1].style, 20);

}

else if(type == 4)

{

strncpy(buffer, films[j].actor, 20);

strncpy(nextBuffer, films[j+1].actor, 20);

}

if(strcmp(nextBuffer, buffer) < 0)

{

filmStruct newFilm = films[j+1];

films[j+1] = films[j];

films[j] = newFilm;

}

}

}

showAllFilms(films);

waitForRequest();

}

void searchChooseTreatment(filmStruct \*films, int type)

{

gotoxy(1, 15);

int index = -1;

if(type == 1 || type == 2)

{

int userEnterInt;

int bufferInt;

scanf("%d", &userEnterInt);

for (int i = 0; i < TOTAL\_FILMS\_COUNT; i++)

{

if(type == 1)

bufferInt = films[i].length;

else

bufferInt = films[i].year;

if (bufferInt == userEnterInt)

{

index = i;

break;

}

}

}

else

{

char userEnter[20];

char buffer[20];

scanf("%s", userEnter);

for (int i = 0; i < TOTAL\_FILMS\_COUNT; i++)

{

if(type == 0)

strncpy(buffer, films[i].name, 20);

else if(type == 3)

strncpy(buffer, films[i].style, 20);

else if(type == 4)

strncpy(buffer, films[i].actor, 20);

if (strcmp(buffer, userEnter) == 0)

{

index = i;

break;

}

}

}

if (index == -1)

cprintf("Не удалось найти");

else

cprintf("%i %s %d %d %s %s", index, films[index].name, films[index].length, films[index].year,

films[index].style, films[index].actor);

waitForRequest();

}

void repaintFrame(char item[5][100], int currentType)

{

textbackground(BLACK);

clrscr();

textbackground(BLUE);

textcolor(WHITE);

for (int i = 0; i < 5; i++)

{

gotoxy(1, i + 1);

cputs(item[i]);

}

textbackground(WHITE);

textcolor(BLUE);

gotoxy(1, currentType + 1);

cputs(item[currentType]);

gotoxy(1, 20);

cprintf("Выход в меню <ESC>");

}