Министерство образования Республики Беларусь

Учреждение образования

«Гомельский государственный университет   
имени Франциска Скорины»

Факультет физики и

информационных технологий

Кафедра общей физики

Разработка программы учета продаж театральных билетов

Курсовая работа

по учебной дисциплине

"Основы алгоритмизации и программирования"

Исполнитель

студент группы МС-12 Романович А. С.

Руководитель

ст. преподаватель Побияха А. С.

Гомель 2021

**РЕФЕРАТ**

**Курсовой проект :** 32 страницы, 29 рисунков.

**Ключевые слова :** Си, База данных, разработка программы, функции, проектирование.

**Цель курсовой работы :** Целью курсовой работы является разработка программы на С для работы театрами.

**Метод исследования** : теоретический метод, эксперимент.

**Задачи курсовой работы :**

1. Анализ предметной области;
2. Разработка сценария работы программы;
3. Анализ исходных данных для проектирования программы;
4. Разработка программы;
5. Тестирование работы программы;

**Вывод :** В ходе данного курсового проекта была спроектирована и разработана программа для работы с театрами.

**Содержание**

Введение……………………………………………………………4

1. Язык программирования С…………………………………….5
   1. Особенности языка С…………………………………5
   2. Основные проблемы языка С………………………...6
   3. Связь языка Си с другими языками программирования..7
2. Проектирование программы……………………………………8
   1. Условие…………………………………………………8
   2. Требование……………………………………………...8
   3. Конструирование программы………………………...10
   4. Выбор способа организации данных…………………11
   5. Схемы…………………………………………………..11
3. Код и работы программы……………………………………….13

Заключение………………………………………………………….20

Приложение…………………………………………………………21

# Введение

Целью курсовой работы является получение студентами теоретических

знаний и практических навыков по конструированию и реализации программ

на языке высокого уровня в рамках процедурной парадигмы.

На уровне конструирования задачей курсовой работы является освоение

технологии нисходящего проектирования и принципов методологии

структурного программирования. Результат этапа конструирования

представляется в форме графических схем алгоритмов.

На уровне программирования задачей курсовой работы является

реализация графических схем алгоритмов на языке высокого уровня с

соблюдением правил соглашения о коде (Code Convention). Результат этапа

программирования представляется в форме проекта в среде разработки

приложений на языке высокого уровня (например, Microsoft Visual Studio при

использовании языка программирования С).

## 1 Язык программирования С

Язык программирования Си — универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней. Большинство программистов предпочитают использовать язык Си для серьезных разработок потому, что их привлекают такие особенности языка, как свобода выражения мыслей, мобильность и чрезвычайная доступность.

## 1.1 Особенности языка С

C является языком функций, типов данных, операторов присваивания и управления последовательностью вычислений. Программируя на C, вы осуществляете обращение к функциям, и большинство функций возвращают некоторые значения. Значение, возвращаемое функцией, будь то значение переменной или константа, может использоваться в операторе присваивания, который изменяет значение другой переменной.

Дополненный операторами управления последовательностью вычислений (while, for, do, switch), C превращается в язык высокого уровня, способствующий хорошему стилю программирования. C имеет небольшой набор типов данных: целые числа, числа с плавающей запятой, битовые поля и перечислимый тип. В языке СC вы можете описать переменную типа указатель, который связывается с объектом, принадлежащим к любому типу данных. Адресная арифметика языка C является чувствительной к типу данных того объекта, с которым связан используемый указатель.

Разрешены также указатели к функциям. Вы можете расширить список типов данных путем создания структур с иерархической зависимостью входящих в него типов данных. Каждый тип данных может принадлежать либо к основному типу, либо к ранее описанному структурному типу. Объединения напоминают структуры, но определяют различные виды иерархических зависимостей, в которых данные разных типов располагаются в памяти.

Допустимо описание массивов данных различных типов, включая структуры и объединения. Массивы могут быть многомерными. Функции C являются рекурсивными по умолчанию. Вы можете, правда, создать функцию, которая не будет рекурсивной, но сам язык по своей природе стремится поддерживать рекурсивность и требует минимальных усилий при программировании рекурсий.

Программа функции на языке C разбивается на блоки, в каждом из которых могут быть определены свои собственные локальные переменные. Блоки могут выбираться для исполнения по результату выполнения оператора управления последовательностью вычислений. Блоки могут быть вложенными друг в друга. Переменные и функции могут быть глобальными для программы, глобальными для исходного модуля или локальными для блока, в котором они описаны. Локальные переменные могут быть описаны таким образом, что они будут сохранять свои значения при всех обращениях внутри данного блока (статические переменные) или же будут восприниматься как новые объекты при каждом обращении (автоматические переменные).

Cи позволяет создавать программу в виде нескольких исходных модулей, которые будут транслироваться независимо. Перемещаемые объектные модули, соответствующие исходным модулям, компонуются в единый загрузочный модуль. Эта особенность позволяет компилятору поддерживать объектные библиотеки многократно используемых функций и создавать большие программы из множества небольших исходных модулей.

В языке C нет операторов ввода/вывода, весь ввод/вывод выполняется с помощью функций. Вследствие этой особенности языка C разработана стандартная библиотека функций. Существование этого стандарта и составляет главную привлекательность языка C, ибо делает программы на C переносимыми.

## 1.2 Основные проблемы языка Си

Язык весьма сложен и наполнен опасными элементами, которые очень легко использовать неправильно. Своей структурой и правилами он никак не поддерживает программирование, нацеленное на создание надёжного и удобного в сопровождении программного кода, напротив, рождённый в эпоху прямого программирования под различные процессоры, язык способствует написанию небезопасного и запутанности кода. Многие профессиональные программисты склонны считать, что язык Си — мощный инструмент для создания элегантных программ, но в то же время с его помощью можно создавать крайне некачественные решения.

Из-за различных допущений в языке программы могут компилироваться с множественными ошибками, что часто приводит к непредсказуемому поведению программы. Современные компиляторы предоставляют опции для статического анализа кода, но даже они не способны выявить все возможные ошибки. Результатом неграмотного программирования на Си могут стать уязвимости программного обеспечения, что может сказаться на безопасности его использования.

У Си высокий порог вхождения. Спецификация его занимает более 500 страниц текста, которые необходимо изучить полностью, так как для создания безошибочного и качественного кода приходится учитывать многие неочевидные особенности языка. Например, автоматическое приведение операндов целочисленных выражений к типу int может давать трудно предсказуемые результаты при использовании бинарных операторов:

Недостаточное понимание подобных нюансов может приводить к появлению многочисленных ошибок и уязвимостей. Ещё одним фактором, увеличивающим сложность освоения Си, является отсутствие обратной связи от компилятора: язык даёт программисту полную свободу действий и позволяет компилировать программы с явными логическими ошибками. Всё это затрудняет использование Си в обучении в качестве первого языка программирования

Наконец, за более чем 40 лет существования язык успел несколько устареть, и в нём достаточно проблематично использовать многие современные приёмы и парадигмы программирования.

**1.3 Связь языка Си с другими языками программирования**

Язык С был и остаётся одним из самых распространённых языков программирования в течение более чем сорока лет. Естественно, что его влияние можно проследить в той или иной мере во многих более поздних языках. Тем не менее среди языков, достигших определённого распространения, прямых потомков у С немного.

Часть языков-потомков надстраивает С дополнительными средствами и механизмами, добавляющими поддержку новых парадигм программирования (ООП, функциональное программирование, обобщённое программирование и пр.). К таким языкам относятся, прежде всего, C++ и Objective-C, а опосредованно — их потомки Swift и D. Также известны попытки улучшить С, исправив его наиболее существенные недостатки, но сохранив его привлекательные черты. Среди них можно упомянуть исследовательский язык Cyclone (и его потомок Rust). Иногда оба направления развития объединяются в одном языке, примером может служить Go.

Отдельно необходимо упомянуть о целой группе языков, которые в большей или меньшей мере унаследовали базовый синтаксис Си (использование фигурных скобок в качестве ограничителей блоков кода, описание переменных, характерные формы операторов for, while, if, switch с параметрами в скобках, комбинированные операции ++, --, +=, -= и другие), из-за чего программы на этих языках имеют характерный внешний вид, ассоциирующийся именно с С. Это такие языки как Java, JavaScript, PHP, Perl, AWK, C#. В действительности структура и семантика этих языков сильно отличается от С, и обычно они предназначены для тех сфер применения, где оригинальный С никогда не использовался.

**2 Разработка программы**

**2.1 Условие**

Разработка программы учета продаж театральных билетов. Сведения

о продажах билетной кассы содержат следующую информацию: дата,

наименование театра, наименование спектакля, количество проданных

билетов.

Индивидуальное задание: вывести список самых популярных театров в

порядке убывания в указанный месяц. Вывести список самых популярных

спектаклей в порядке убывания в указанный месяц. Месяц вводится с

клавиатуры.

Общее для всех вариантов задание: реализовать авторизацию для

входа в систему, функционал администратора и функционал пользователя.

**2.2 Требования**

Первым этапом работы программы является авторизация –

предоставление прав доступа. В рамках данного этапа необходимо считать

данные из файла с учетными записями пользователей следующего вида:

login; password; role (данное поле служит для разделения в правах

администраторов и пользователей).

После ввода пользователем своих персональных данных (логина и

пароля) и сверки со считанной из файла информацией необходимо

предусмотреть возможность входа: в качестве администратора; в качестве

пользователя.

Регистрация новых пользователей осуществляется администратором в

режиме работы с учетными записями пользователей (т. е. администратор сам

создает для пользователей аккаунты).

Вторым этапом работы программы является собственно работа с

данными, которая становится доступной только после прохождения

авторизации. Данные хранятся в отдельном файле и имеют вид, описанный

подробно в каждом варианте к курсовой работе.

Для работы с данными должны быть предусмотрены два

функциональных модуля: модуль администратора и модуль пользователя.

Модуль администратора включает следующие подмодули (с указанием

функциональных возможностей):

1. Управление учетными записями пользователей: просмотр всех

учетных записей; добавление новой учетной записи; редактирование учетной

записи; удаление учетной записи.

2. Работа с данными:

а) режим редактирования: просмотр всех данных; добавление

новой записи; удаление записи; редактирование записи;

б) режим обработки данных: выполнение индивидуального

задания; поиск данных; сортировка.

Модуль пользователя включает подмодуль работы с данными со

следующими функциональными возможностями: просмотр всех данных;

выполнение индивидуального задания; поиск данных; сортировка.

Для реализации перечисленных модулей/подмодулей необходимо

создавать меню с соответствующими пунктами.

В курсовой работе необходимо предусмотреть:

1. Обработку исключительных ситуаций: введенные пользователем

данные не соответствуют формату поля (например, символы в числовом

поле); введенные пользователем данные нелогичны (например,

отрицательная цена товара); файл с данными для чтения не существует;

ничего не найдено по результатам поиска; номер удаляемой записи выходит

за пределы массива/вектора.

2. Возможность возврата назад (навигация).

3. Запрос на выполнение необратимых действий, а именно

подтверждение удаления вида «Вы действительно хотите удалить файл

(запись)?».

4. Обратную связь с пользователем, например, вывод сообщения об

успешности удаления/редактирования записи и т. д.

Требования к программной реализации.

1. Все переменные и константы должны иметь осмысленные имена в

рамках тематики варианта курсовой работы. Переменным рекомендуется

присваивать имена, состоящие из букв нижнего регистра; для формирования

составного имени используется нижнее подчеркивание (например,

number\_of\_students) или «верблюжья нотация» (например, flagExit).

2. Имена функций должны быть осмысленными, начинаться с буквы

нижнего регистра, строиться по принципу глагол + существительное

(например, addAccount, findStudentBySurname). Если функция выполняет

проверку и возвращает результат типа bool, то ее название должно

начинаться с глагола is (например, isNumberNumeric, isLoginUnique).

3. Не допускается использование оператора прерывания goto.

4. Код не должен содержать неименованных числовых констант

(«магических» чисел), неименованных строковых констант (например, имен

файлов и др.). Подобного рода информацию следует представлять, как

глобальные константы. По правилам качественного стиля программирования

тексты всех информационных сообщений, выводимых пользователю в ответ

на его действия, также оформляются как константы.

5. Код необходимо комментировать (как минимум в части объявления

структур, массивов/векторов, прототипов функций, нетривиальной логики).

6. Код не должен дублироваться – для этого существуют функции.

7. Одна функция решает только одну задачу (например, не допускается в

одной функции считывать данные из файла и выводить их на консоль – это

две разные функции). При этом внутри функции возможен вызов других

функций.

8. Выполнение операций чтения/записи в файл должно быть сведено к

минимуму (т. е. после однократной выгрузки данных из файла в

массив/вектор дальнейшая работа ведется с этим массивом/вектором, а не

происходит многократное считывание данных из файла в каждой функции).

9. Следует избегать глубокой вложенности условных и циклических

конструкций: вложенность блоков должна быть не более трех.

10. Следует избегать длинных функций: текст функции должен

умещаться на один экран (размер текста не должен превышать 25–50 строк).

Выбор способов организации данных.

1. В качестве выбора способа описания входных данных приводится

описание следующих типов struct (с указанием конкретных полей): для

учетных записей пользователей; для данных.

2. В качестве способа объединения входных данных указывается:

использование массивов (статически или динамически создаваемых) или

векторов; их выбранная область видимости (локальные или глобальные).

Разработка перечня пользовательских функций программы

подразумевает перечисление прототипов функций, необходимых для

реализации программы, и краткие комментарии к ним.

Разработка алгоритмов работы программы включает схемы алгоритмов

(с кратким словесным описанием их работы в тексте пояснительной записки)

для главной функции и двух любых других пользовательских функций.

Алгоритм должен быть оформлен согласно ГОСТ 19.701-90 Схемы

алгоритмов, программ, данных и систем. При разработке алгоритмов

рекомендуется использовать профессиональные графические редакторы

диаграмм и блок-схем, например, Microsoft Visio. Описание работы

программы подразумевает краткое словесное описание работы программы со скриншотами консоли.

**2.3 Конструирование программы**

Реализация программы будет осуществляться на языке С в IDE среде Embarcadero Dev-C++.

Разработка структуры программы

Согласно требованиям к программе, необходимо наличие исполняемой программы, которая работает с базой пользователей программы и базой путевок. Следовательно, с точки зрения верхней архитектуры программы, можно выделить два основных модуля: модуль работы с базой пользователей и модуль работы с базой театров. На рисунках 1 и 2 показаны основные модули программы

Функция аутентификации пользователя заключается в проверке существования данных пользователей введённого логина и соответствующего ему пароля. Авторизация пользователя подразумевает получение его роли из файла данных и предоставление ему соответствующих привилегий. Задачи аутентификации пользователя и запрос его роли можно реализовать в отдельном модуле либо непосредственно в модуле управления данными пользователей. После успешной авторизации пользователя создаётся пользовательская сессия с соответствующими привилегиями, согласно роли пользователя. Сессия администратора имеет полный доступ к модулям управления обеими базами данных, а сессия пользователя, в свою очередь, имеет доступ только к модулю управления базой данных пациентов в режиме «только для чтения».

Программа подразумевает наличие консольного пользовательского интерфейса.

**2.4 Выбор способа организации данных**

В качестве БД был выбран текстовый файл.

Программа обслуживает данные, хранимые в двух базах данных: базе данных пользователей и базе данных автовокзала. Эти базы имеют различные схемы данных. Схема базы данных пользователей Для базы данных пользователей требуется хранить имя пользователя, пароль и роль пользователя. Схема базы данных пользователей показана на рисунке 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **login** | **Password** | **Role** |

Рисунок 1 - Схема базы данных пользователей

Схема базы данных театров

База данных билетной кассы, согласно требованиям, должна содержать следующую информацию:

Дату;

Наименование театра;

Наименование спектакля;

Количество проданный билетов;

Схема базы данных показана на рисунке 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **date** | **nameT** | **nameS** | **tickets** |

Рисунок 2 - Схема базы данных театра

**2.5 Схемы**

В ходе разработки были составлены следующие схемы :

1)Схема работы пользователя :

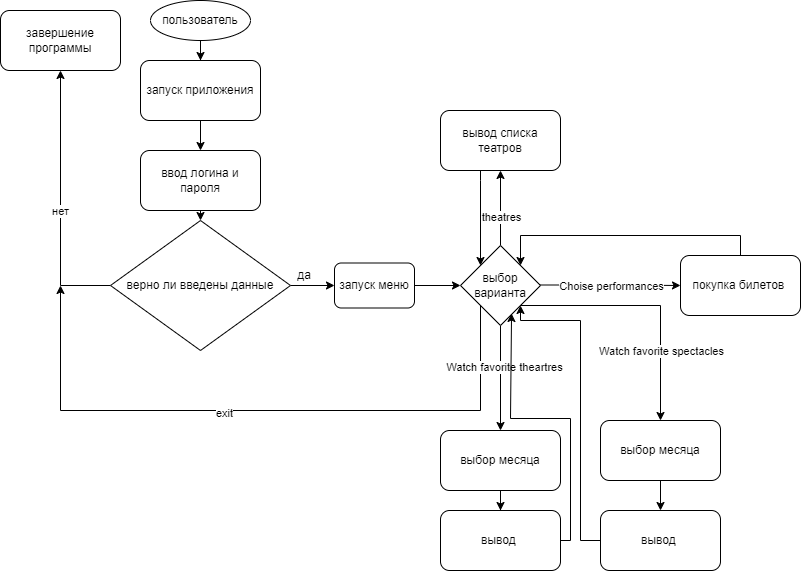


Рисунок 3 - Схема работы программы для пользователя

2)Схема работы Администратора

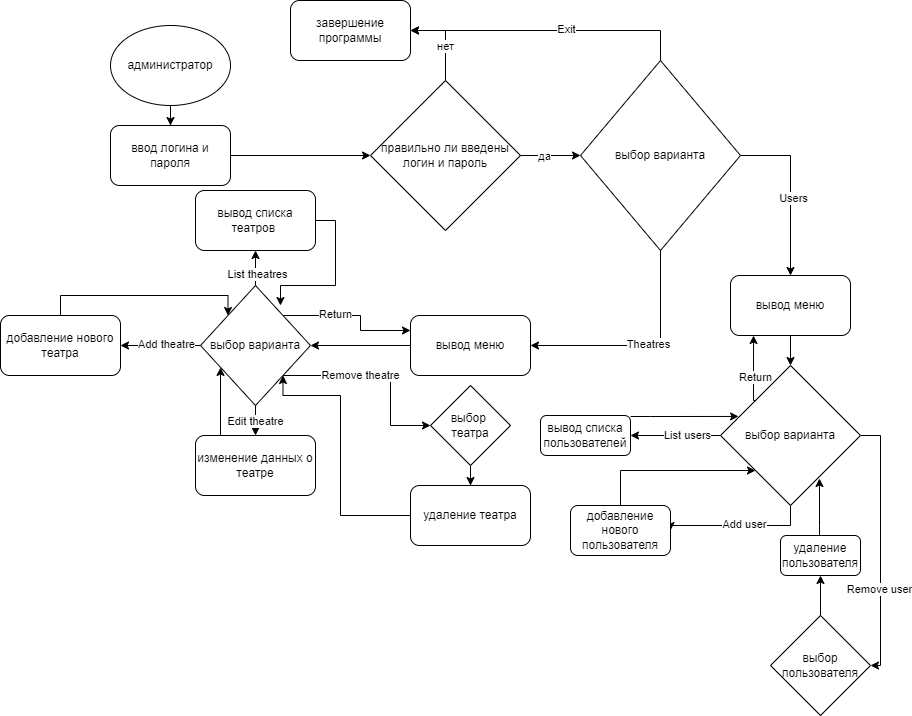


Рисунок 4 - Схема работы программы для администратора

**3 Код и работа программы**

Данная программа разрабатывалась функциями и структурами

Основное тело программы :

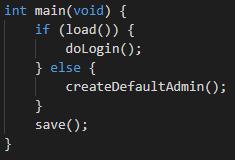


Рисунок 5 – Код основного тела программы

Функция отвечающая за загрузку данных из базы данных :

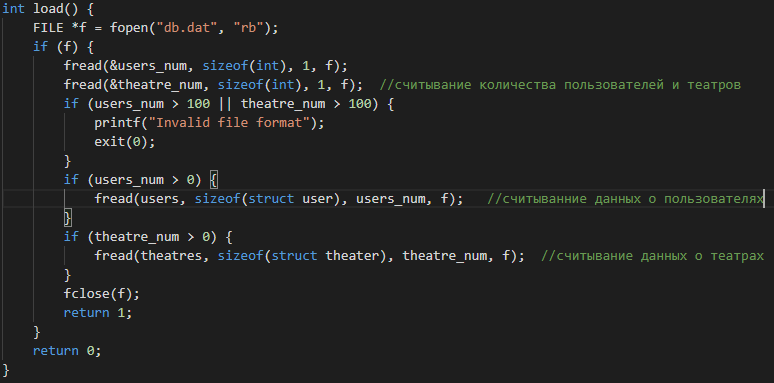


Рисунок 6 – Код загрузки данных

Функция загружающая все изменения данных в файл :

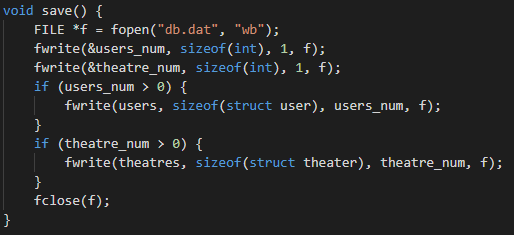


Рисунок 7 – Код сохранения данных

Структура включающая в себя данные о конкретном театре:

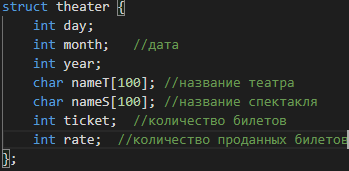


Рисунок 8 – Структура театров

Структура данных о пользователях :

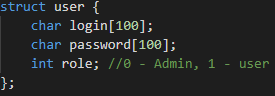


Рисунок 9 – Структура данных пользователей

Функция проверки логина и пароля и авторизации :

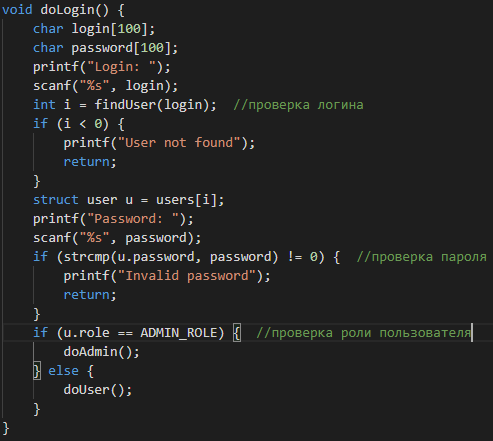


Рисунок 10 – Основная часть функции проверки логина и пароля

****

Рисунок 11 - Работа функции проверки логина и пароля.

**Рассмотрим работу программы со стороны администратора :**

Функция меню работы программы с администратором :

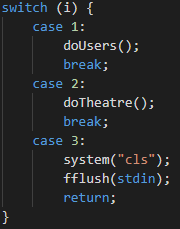


Рисунок 12 – Основная часть функции меню администратора

Функция switch отвечает за перемещение администратора по программе

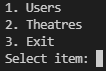


Рисунок 13 - Работа функции меню работы программы с администартором

Функция меню администратора работы с пользователями :

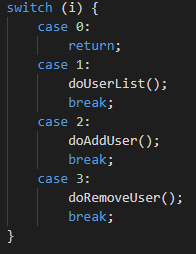


Рисунок 14 – Основная часть функции вывода меню администратора для работы с пользователями

Здесь так же switch отвечает за дальнейшее взаимодействие с программой

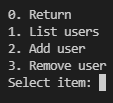


Рисунок 15 - Работа функции меню администратора работы с пользователями

Функция выводящая всех пользователей и структурирующая их :

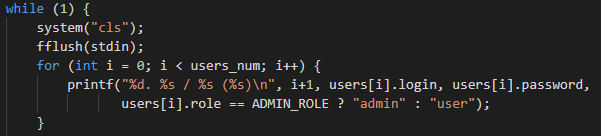


Рисунок 16 – Основная часть функции вывода списка пользователей



Рисунок 17 - Работа функции, которая выводит всех пользователей на экран и структурирует их :

Функция добавления нового пользователя :

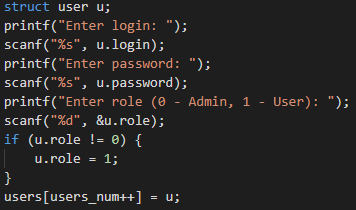


Рисунок 18 – Основная часть функции добавления нового пользователя

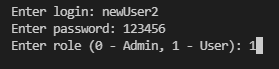


Рисунок 19 - Работы функции дабовления нового пользователя

Функция удаления пользователя:

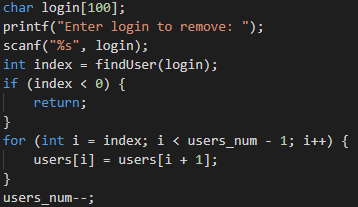


Рисунок 20 – Основная часть функции удаления пользователя



Рисунок 21 - Работа функции удаления пользователя

Функция работы с театрами для администратора :

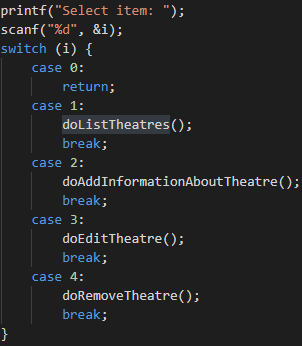


Рисунок 22 – Основная часть функции администратора для работы с пользователями

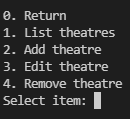


Рисунок 23 - Работа функции меню работы администратора с театрами

Функция выводит для вывода всю информацию про театры на экран:

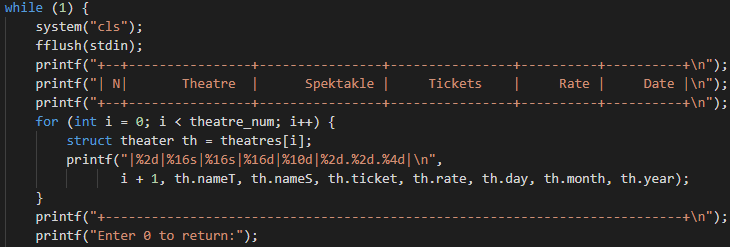
****

Рисунок 24 – Основная часть функции вывода списка театров

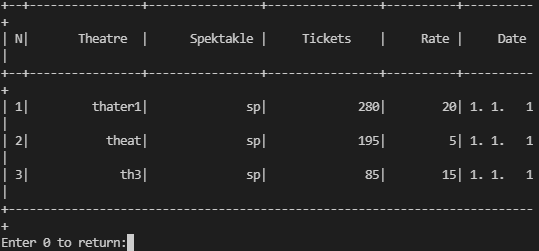


Рисунок 25 – Работа функции вывода списка театров на экран

Остальные функции работы с театрами имеют схожее строение с функциями работы с пользователями.

**Работа программы со стороны пользователя** **:**

Вывод меню работы приложения для пользователя :

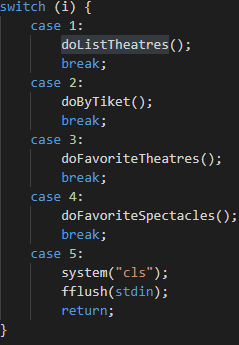


Рисунок 26 – Основная часть функции вывода меню пользователя

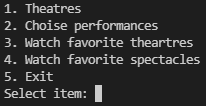


Рисунок 27 – Работа функции вывода меню на экран

Функция покупки билета :

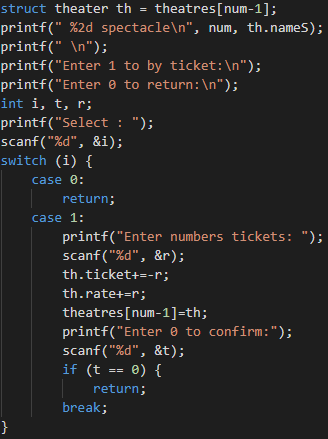


Рисунок 28 – Основная часть функции пользователя для покупки билета

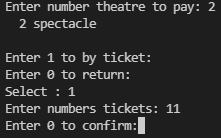


Рисунок 29 – Работа функции покупки билетов

**Заключение**

В результате выполнения курсового проекта было разработано приложение для продажи билетов в театр. Помимо продажи билетов в театр, доступен просмотр статистики и поиск данных для более доступной организации информации. Доступен вход в систему в качестве администратора. В этом режиме у пользователя появляется возможность не только редактировать отдельные поля классов, но и добавлять новые записи, удалять ненужную или же по ошибке добавленную информацию.

Незаменимой вещью в ходе выполнения курсового проекта оказался инструментарий языка по работе с текстовыми и бинарными файлами. В данной работе был выбран именно второй тип. Благодаря использованию файлов появляется возможность сохранять информацию после завершения сеанса и загружать сохраненные данные при запуске приложения.

**Приложение**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

const int ADMIN\_ROLE = 0;

const int USER\_ROLE = 1;

struct user {

    char login[100];

    char password[100];

    int role;

};

struct theater {

    int day;

    int month;

    int year;

    char nameT[100];

    char nameS[100];

    int ticket;

    int rate;

};

struct user users[100];

int users\_num = 0;

struct theater theatres[100];

int theatre\_num = 0;

int load() {

    FILE \*f = fopen("db.dat", "rb");

    if (f) {

        fread(&users\_num, sizeof(int), 1, f);

        fread(&theatre\_num, sizeof(int), 1, f);

        if (users\_num > 100 || theatre\_num > 100) {

            printf("Invalid file format");

            exit(0);

        }

        if (users\_num > 0) {

            fread(users, sizeof(struct user), users\_num, f);

        }

        if (theatre\_num > 0) {

            fread(theatres, sizeof(struct theater), theatre\_num, f);

        }

        fclose(f);

        return 1;

    }

    return 0;

}

void save() {

    FILE \*f = fopen("db.dat", "wb");

    fwrite(&users\_num, sizeof(int), 1, f);

    fwrite(&theatre\_num, sizeof(int), 1, f);

    if (users\_num > 0) {

        fwrite(users, sizeof(struct user), users\_num, f);

    }

    if (theatre\_num > 0) {

        fwrite(theatres, sizeof(struct theater), theatre\_num, f);

    }

    fclose(f);

}

int findUser(char login[10]) {

    for (int i = 0; i < users\_num; i++) {

        if (strcmp(users[i].login, login) == 0) {

            return i;

        }

    }

    return -1;

}

void doUserList() {

    while (1) {

        system("cls");

        fflush(stdin);

        for (int i = 0; i < users\_num; i++) {

            printf("%d. %s / %s (%s)\n", i+1, users[i].login, users[i].password,

                   users[i].role == ADMIN\_ROLE ? "admin" : "user");

        }

        printf("Enter 0 to return:");

        int i;

        scanf("%d", &i);

        if (i == 0) {

            return;

        }

    }

}

void doAddUser() {

    system("cls");

    fflush(stdin);//clear input

    struct user u;

    printf("Enter login: ");

    scanf("%s", u.login);

    printf("Enter password: ");

    scanf("%s", u.password);

    printf("Enter role (0 - Admin, 1 - User): ");

    scanf("%d", &u.role);

    if (u.role != 0) {

        u.role = 1;

    }

    users[users\_num++] = u;

}

void doRemoveUser() {

    system("cls");

    fflush(stdin);

    char login[100];

    printf("Enter login to remove: ");

    scanf("%s", login);

    int index = findUser(login);

    if (index < 0) {

        return;

    }

    for (int i = index; i < users\_num - 1; i++) {

        users[i] = users[i + 1];

    }

    users\_num--;

}

void doUsers() {

    while (1) {

        system("cls");

        fflush(stdin);

        printf("0. Return\n");

        printf("1. List users\n");

        printf("2. Add user\n");

        printf("3. Remove user\n");

        int i;

        printf("Select item: ");

        scanf("%d", &i);

        switch (i) {

            case 0:

                return;

            case 1:

                doUserList();

                break;

            case 2:

                doAddUser();

                break;

            case 3:

                doRemoveUser();

                break;

        }

    }

}

void doListTheatres() {

    while (1) {

        system("cls");

        fflush(stdin);

        printf("+--+----------------+----------------+----------------+----------+----------+\n");

        printf("| N|       Theatre  |      Spektakle |     Tickets    |     Rate |     Date |\n");

        printf("+--+----------------+----------------+----------------+----------+----------+\n");

        for (int i = 0; i < theatre\_num; i++) {

            struct theater th = theatres[i];

            printf("|%2d|%16s|%16s|%16d|%10d|%2d.%2d.%4d|\n",

                   i + 1, th.nameT, th.nameS, th.ticket, th.rate, th.day, th.month, th.year);

        }

        printf("+---------------------------------------------------------------------------+\n");

        printf("Enter 0 to return:");

        int i;

        scanf("%d", &i);

        if (i == 0) {

            return;

        }

    }

}

void doAddInformationAboutTheatre() {

    system("cls");

    fflush(stdin);

    struct theater th;

    printf("Enter name theatre: ");

    scanf("%s", th.nameT);

    printf("Enter name spectakle: ");

    scanf("%s", th.nameS);

    printf("Enter how much tickets: ");

    scanf("%d", &th.ticket);

    th.rate = 0;

    printf("Enter date performance \n");

    printf("Enter day: ");

    scanf("%d", &th.day);

    printf("Enter month: ");

    scanf("%d", &th.month);

    printf("Enter year: ");

    scanf("%d", &th.year);

    theatres[theatre\_num++] = th;

}

void doEditTheatre() {

    system("cls");

    fflush(stdin);

    int num;

    printf("Enter theatre number to edit: ");

    scanf("%d", &num);

    if (num > theatre\_num) {

        return;

    }

    struct theater th;

    printf("Enter name theatre: ");

    scanf("%s", th.nameT);

    printf("Enter name spectakle: ");

    scanf("%s", th.nameS);

    printf("Enter how much tickets: ");

    scanf("%d", &th.ticket);

    printf("Enter date performance \n");

    printf("Enter day: ");

    scanf("%d", &th.day);

    printf("Enter month: ");

    scanf("%d", &th.month);

    printf("Enter year: ");

    scanf("%d", &th.year);

    theatres[num - 1] = th;

}

void doRemoveTheatre() {

    system("cls");

    fflush(stdin);

    int num;

    printf("Enter theatre number to remove: ");

    scanf("%d", &num);

    if (num > theatre\_num) {

        return;

    }

    int index = num - 1;

    for (int i = index; i < theatre\_num - 1; i++) {

        theatres[i] = theatres[i + 1];

    }

    theatre\_num--;

}

void doTheatre() {

    while (1) {

        system("cls");

        fflush(stdin);

        printf("0. Return\n");

        printf("1. List theatres\n");

        printf("2. Add theatre\n");

        printf("3. Edit theatre\n");

        printf("4. Remove theatre\n");

        int i;

        printf("Select item: ");

        scanf("%d", &i);

        switch (i) {

            case 0:

                return;

            case 1:

                doListTheatres();

                break;

            case 2:

                doAddInformationAboutTheatre();

                break;

            case 3:

                doEditTheatre();

                break;

            case 4:

                doRemoveTheatre();

                break;

        }

    }

}

void doByTiket() {

    system("cls");

    fflush(stdin);

    int num;

    printf("Enter number theatre to pay: ");

    scanf("%d", &num);

    if (num > theatre\_num) {

        return;

    }

        struct theater th = theatres[num-1];

        printf(" %2d spectacle\n", num, th.nameS);

        printf(" \n");

        printf("Enter 1 to by ticket:\n");

        printf("Enter 0 to return:\n");

        int i, t, r;

        printf("Select : ");

        scanf("%d", &i);

        switch (i) {

            case 0:

                return;

            case 1:

                printf("Enter numbers tickets: ");

                scanf("%d", &r);

                th.ticket+=-r;

                th.rate+=r;

                theatres[num-1]=th;

                printf("Enter 0 to confirm:");

                scanf("%d", &t);

                if (t == 0) {

                    return;

                break;

        }

    }

}

int sort(const void \*a, const void \*b) {

    return ((struct theater \*) b)->rate - ((struct theater \*) a)->rate;

}

void doFavoriteSpectacles() {

    while (1) {

        system("cls");

        fflush(stdin);

        printf ("Enter the month in which the performance took place:\n");

        int g;

        printf("Select item: ");

        scanf("%d", &g);

        struct theater th;

        if (th.month == g) {

        printf("+--+----------------+----------------+----------+\n");

        printf("| N|       Theatre  |      Spektakle |   Rate|\n");

        printf("+--+----------------+----------------+----------+\n");

        struct theater sortedTheatres[theatre\_num];

        qsort (sortedTheatres, theatre\_num, sizeof(struct theater), sort);

        for (int i = 0; i < theatre\_num && i < 15; i++) {

            struct theater th = sortedTheatres[i];

            printf("|%2d|%16s|%16s|%10d|\n",

                    i + 1, th.nameT, th.nameS, th.rate);

        }

        int i;

        printf("+------------------------------------+\n");

        }

        printf("Enter 0 to return:");

        scanf("%d", &i);

        if (i == 0) {

            return;

        }

    }

}

void doFavoriteTheatres() {

    while (1) {

        system("cls");

        fflush(stdin);

        printf ("Enter the month in which the performance took place:\n");

        int g;

        printf("Select item: ");

        scanf("%d", &g);

        struct theater th;

        if (th.month == g) {

            char name1[100];

            char name2[100];

            struct theater th = theatres[i];

            for (int i = 0; i<theatre\_num; i++) {

                struct theater th = theatres[i];

                name1[] = th.nameT;

                struct theater th = theatres[i+1];

                name2[] = th.nameT;

                if (name1 == name2) {

                    th.rate += th.rate;

                }

            }

        printf("+--+----------------+----------------+----------+\n");

        printf("| N|       Theatre  |      Spektakle |   Rate|\n");

        printf("+--+----------------+----------------+----------+\n");

        struct theater sortedTheatres[theatre\_num];

        qsort (sortedTheatres, theatre\_num, sizeof(struct theater), sort);

        for (int i = 0; i < theatre\_num && i < 15; i++) {

            struct theater th = sortedTheatres[i];

            printf("|%2d|%16s|%16s|%10d|\n",

                    i + 1, th.nameT, th.nameS, th.rate);

        }

        int i;

        printf("+------------------------------------+\n");

        }

        printf("Enter 0 to return:");

        scanf("%d", &i);

        if (i == 0) {

            return;

        }

    }

}

void doAdmin() {

    while (1) {

        system("cls");

        fflush(stdin);

        printf("1. Users\n");

        printf("2. Theatres\n");

        printf("3. Exit\n");

        int i;

        printf("Select item: ");

        scanf("%d", &i);

        switch (i) {

            case 1:

                doUsers();

                break;

            case 2:

                doTheatre();

                break;

            case 3:

                system("cls");

                fflush(stdin);

                return;

        }

    }

}

void doUser() {

    while (1) {

        system("cls");

        fflush(stdin);

        printf("1. Theatres\n");

        printf("2. Choise performances\n");

        printf("3. Watch favorite theartres\n");

        printf("4. Watch favorite spectacles\n");

        printf("5. Exit\n");

        int i;

        printf("Select item: ");

        scanf("%d", &i);

        switch (i) {

            case 1:

                doListTheatres();

                break;

            case 2:

                doByTiket();

                break;

            case 3:

                doFavoriteTheatres();

                break;

            case 4:

                doFavoriteSpectacles();

                break;

            case 5:

                system("cls");

                fflush(stdin);

                return;

        }

    }

}

void doLogin() {

    char login[100];

    char password[100];

    printf("Login: ");

    scanf("%s", login);

    int i = findUser(login);

    if (i < 0) {

        printf("User not found");

        return;

    }

    struct user u = users[i];

    printf("Password: ");

    scanf("%s", password);

    if (strcmp(u.password, password) != 0) {

        printf("Invalid password");

        return;

    }

    if (u.role == ADMIN\_ROLE) {

        doAdmin();

    } else {

        doUser();

    }

}

void createDefaultAdmin() {

    system("cls");

    fflush(stdin);

    struct user u = {"admin", "pass", ADMIN\_ROLE};

    users[users\_num++] = u;

    printf("New database with default admin created. Use 'admin' for login and 'pass' for password\n");

}

int main(void) {

    if (load()) {

        doLogin();

    } else {

        createDefaultAdmin();

    }

    save();

}