Министерство образования Республики Беларусь

Учреждение образования

Белорусский государственный университет

информатики и радиоэлектроники

Институт информационных технологий

Курсовая работа по предмету «Основы конструирования программ»

Разработал: Дедюль А.Н.

студент группы 980971

Проверил: Рябычина О.П.

преподаватель

Минск 2020

Содержание

[1. Требования к программе 4](#_Toc29723577)

[2. Конструирование программы 7](#_Toc29723578)

[2.1 Разработка модульной структуры программы 7](#_Toc29723579)

[2.2 Выбор способа организации данных 7](#_Toc29723580)

[2.3 Разработка перечня пользовательских функций программы 8](#_Toc29723581)

[3. Разработка алгоритмов работы программы 10](#_Toc29723582)

[3.1 Алгоритм функции main 10](#_Toc29723583)

[3.2 Алгоритм функции menuAutorization 10](#_Toc29723584)

[3.3 Алгоритм функции foundForFIO 12](#_Toc29723585)

[4. Описание работы программы 14](#_Toc29723586)

[4.1 Авторизация 14](#_Toc29723587)

[4.2 Модуль администратора 14](#_Toc29723588)

[4.3 Модуль пользователя 18](#_Toc29723589)

[4.4 Исключительные ситуации 18](#_Toc29723590)

[Заключение 19](#_Toc29723591)

[Приложение 20](#_Toc29723592)

**1. Требования к программе**

Разработка программы учёта подписчиков на печатную продукцию.

Сведения о подписчиках компании имеют следующий вид: Ф.И.О. подписчика; год; месяц; количество дней подписки; оплата за один день.

Индивидуальное задание: для месяца х года у вывести список подписчиков с указанием выплат по подпискам для каждого из них. Вывести общую сумму выплат по подпискам за интересующий месяц. Х, у вводятся с клавиатуры.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя, как минимум три вида поиска, как минимум три вида сортировки.

Исходные данные к курсовой работе:

* язык программирования С++;
* среда разработки Visual Studio 2015;
* вид приложения – консольное;
* парадигма программирования – процедурная;
* способ организации данных – массивы;
* способ хранения данных – текстовые файлы.

Функциональные требования:

* авторизация – предоставление прав доступа. Считать данные из файла с учетными записями пользователей;
* работа с данными, только после прохождения авторизации, данные хранятся в отдельном файле;
* два функциональных модуля: модуль администратора и модуль пользователя.

Модуль администратора включает следующие подмодули:

* управление учетными записями пользователей: просмотр всех учетных записей, добавление новой учетной записи, редактирование учетной записи, удаление учетной записи;
* работа с данными: просмотр всех данных, добавление новой записи, удаление записи, редактирование записи, выполнение индивидуального задания, поиск данных, сортировка.

Модуль пользователя включает подмодуль работы с данными со следующими функциональными возможностями:

* просмотр данных;
* поиск данных;
* сортировка;
* выполнение индивидуального задания.

Так же следует предусмотреть обработку исключительных ситуаций:

* введенные пользователем данные не соответствуют формату поля;
* введенные пользователем данные некорректны;
* файл с данными для чтения не существует;
* ничего не найдено по результатам поиска;
* номер удаляемой записи выходит за пределы массива;
* логин новой учетной записи уже существует;
* возможность возврата назад;
* обратную связь с пользователем.

Требования к программной реализации:

* переменные и константы должны иметь осмысленные имена;
* имена функций должны быть осмысленными и строится по принципу глагол+существительное, если функция выполняет какую-либо проверку и возвращает результат типа bool, то ее название должно начинаться с глагола is;
* не использовать оператор goto;
* код не должен содержать неименованных числовых констант, неименованных строковых констант. Подобного рода информацию следует выносить в глобальные переменные с атрибутом const. Тексты всех информационных сообщений, выводимых пользователю в ответ на его действия, также оформляются как константы;
* код необходимо комментировать;
* код не должен дублироваться;
* одна функция должна решать только одну задачу, при этом внутри функции возможен вызов других функций;
* выполнение операций чтения/записи в файл должно быть сведено к минимуму;
* следует избегать длинных функций и глубокой вложенности: текст функции должен умещаться на один экран, а вложенность блоков и операторов должна быть не более трёх;
* выносите код логически независимых модулей в отдельные .cpp файлы и подключать их с помощью заголовочных .h файлов.

**2. Конструирование программы**

## 2.1 Разработка модульной структуры программы

Во время конструирования программы, первым этапом был анализ и разработка структура программы. Графическое представление структуры программы представлено на рисунке 2.1.

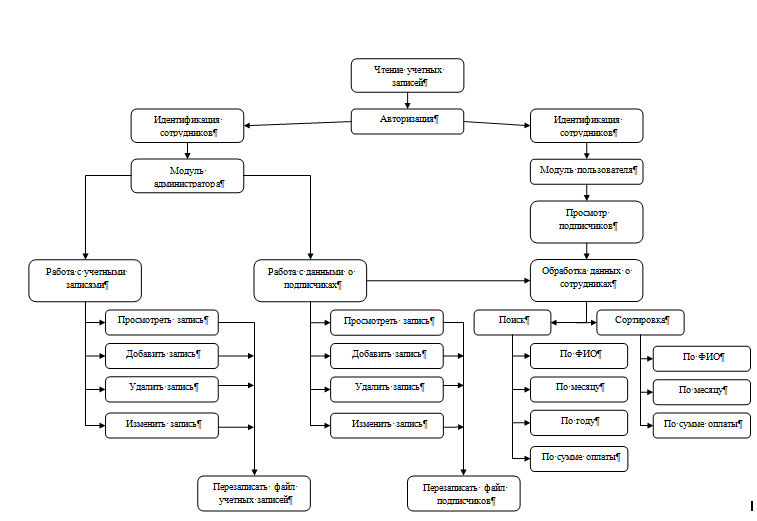


Рисунок 2.1 – Структура программы

Разработанная структура программы, позволила разбить приложение на подмодули и облегчить разработку приложения.

## 2.2 Выбор способа организации данных

Следующий этап при конструировании программы, выбор способа организации данных.

Входные данные:

Для учетных записей пользователей:

* логин, тип string;
* пароль, тип string;
* роль, тип integer.

Для данных о подписчиках:

* ФИО, тип string;
* месяц, тип integer;
* дни, тип integer;
* год, тип integer;
* сумма выплат, тип integer.

Наиболее подходящий способ организации данных – массив. У массива есть некоторые преимущества над вектором.

* лёгкость вычисления адреса элемента по его индексу (поскольку элементы массива располагаются один за другим)
* одинаковое время доступа ко всем элементам
* малый размер элементов: они состоят только из информационного поля.

## 2.3 Разработка перечня пользовательских функций программы

В ходе разработки приложения, были реализованы следующие функции:

* меню пользователя – menu;
* меню администратора – menuAdmin;
* меню учётных записей – menuAccounts;
* меню листов подписчиков – menuList;
* меню листов подписчиков для администратора – menuListbyAdmin;
* использование меню – useMenu;
* использование меню администратора – useMenuAdmin;
* использование меню аккаунтов– useMenuAccounts;
* использование меню листов подписчиков – useMenuList;
* использование дополнительного меню – useMenuTwo;
* использование меню подписчиков для администратора – useMenuListbyAdmin;
* опции листов подписчиков – optionList;
* использование меню опций листов подписчиков – useMenuOptionList;
* меню авторизации– menuAutorization;
* запись в файл аккаунтов – writeFileAccounts;
* запись в конец файла аккаунтов– writeEndFileAccounts;
* чтение файла аккаунтов – readFileAccounts;
* чтение файла листов подписчиков – readFileLists;
* запись в конец файла листов подписчиков – writeEndFileLists;
* запись в файл листов подписчиков – writeFileLists;
* вывод списка аккаунтов на экран – showAccounts;
* добавление учётной записи – addAccount;
* удаление учётной записи – delAccount;
* редактирование учётной записи – updateAccount;
* вывод списка подписчиков – showLists;
* вывод списка найденных подписчиков – showFoundLists;
* добавление подписчика – addLists;
* удаление подписчика – delLists;
* редактирование подписчика – updateLists;
* поиск подписчика – foundList;
* поиск оплаты по подписчикам– foundPay;
* поиск по ФИО – foundForFIO;
* поиск по месяцу – foundForMounth;
* поиск по году – foundForYear;
* поиск по дням– foundForDay;
* поиск по сумме опдаты – foundForPay;
* сортировка по ФИО – SortByFIO;
* сортировка по месяцу – SortByMounth;
* сортировка по сумме оплаты – SortByPay;

Разбиение приложения на функции, позволяет следовать некоторым принципам SOLID.

# 3. Разработка алгоритмов работы программы

## 3.1 Алгоритм функции main

Блок-схема функции main представлена на рисунке 3.1.

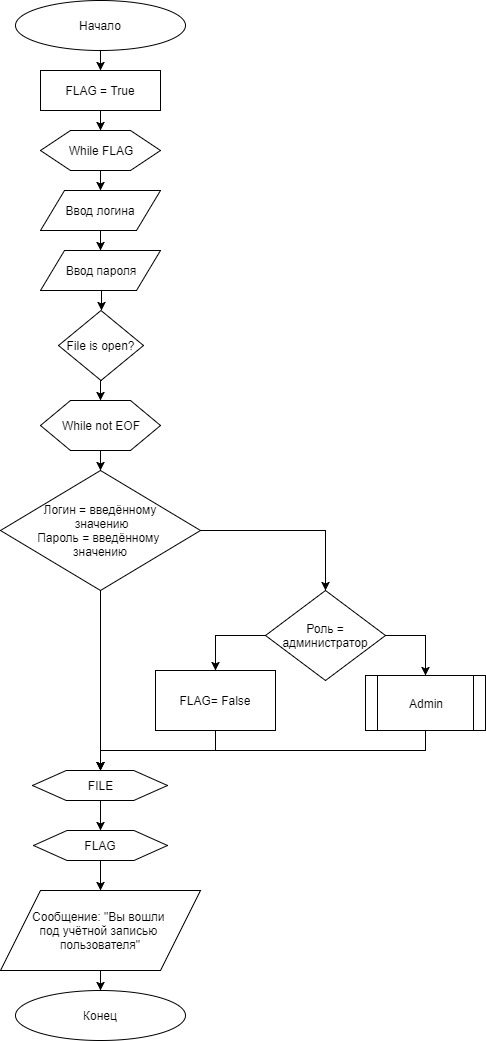


Рисунок 3.1 – Блок-схема функции main

В функции «Настройка консоли» будут установлены кодовые страницы вывода/вывода данных, используемых консолью, а также задана локализация консоли. Далее будет вызвана функция «Меню», в которой будет происходить авторизация пользователя, а после успешной авторизации будут заполнены массивы данных. После будут вызваны функция «Меню», для сотрудника либо же для администратора, в зависимости от введённых при авторизации данных.

## 3.2 Алгоритм функции menuAutorization

Блок-схема функции menuAutorization представлена на рисунке 3.2.



0

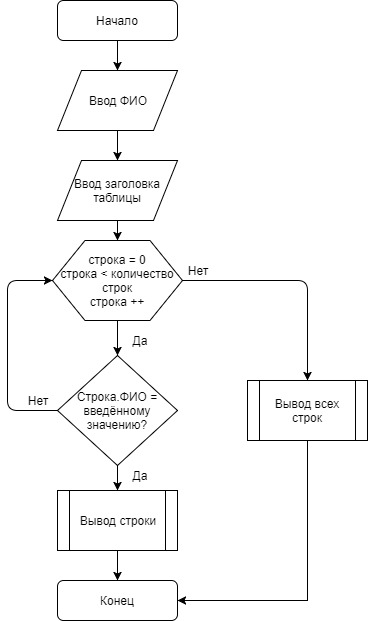
1

Рисунок 3.2 – Блок-схема функции ShowAuthorizationMenu

Функция авторизации. В начале пользователь вводит логин и пароль. Далее идёт проверка на то открыт файл с данными или нет, если файл не открыт, то появляется сообщение о том, что файл недоступен. Если же файл открыт, то алгоритм продолжается. Далее идёт вход в цикл, в котором поочерёдно перебираются все записи в файле. Далее при проверке соответствия введённых с клавиатуры значений логина и пароля и данных из файла, при соответствии осуществляется переход на следующий этап. При несоответствии цикл продолжается. Если логин и пароль соответствуют данным из файла они подвергаются ещё одной проверки, проверки на Роль, если роль учётной записи Администратор, то происходит выход из цикла и вывод меню администратора. Если же роль не соответствует Администратору, то происходит мгновенный выход из цикла и открытие меню пользователя.

## 3.3 Алгоритм функции foundForFIO

Блок-схема функции foundForFIO представлена на рисунке 3.3.



Сообщение: «Ничего не найдено»

Рисунок 3.3 – Блок-схема функции FindEmployeeByLastName

Функция поиска. Пользователю будет предложено ввести значение для поиска сотрудников по ФИО. Затем происходит ввод заголовка таблицы. После этого программа входит в цикл и до тех пор, пока программа не дойдёт до последней строки, будут выполняться следующие действия:

Проверка условия равны ли введённое с клавиатуры значение и значение поля в файле. Если нет, то происходить продвижение по циклу. Если же данные совпадают происходит вывод их на экран.

Пройдя весь путь по циклу и не найдя одинаковых данных, программа, выведет весь список больничных листов.

# 4. Описание работы программы

## 4.1 Авторизация

Первое что пользователь видит при входе в программу это - Меню авторизации, рисунок 4.1.

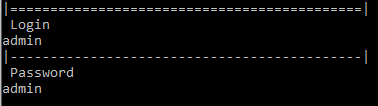


Рисунок 4.1 – Меню авторизации

После успешной авторизации пользователь попадает в меню, в зависимости от того под какой ролью он зашёл. Если пользователь вошёл в программу под профилем «Администратор», то после авторизации ему открывается модуль администратора.

## 4.2 Модуль администратора

После авторизации переходим в меню администратора, рисунок 4.2.

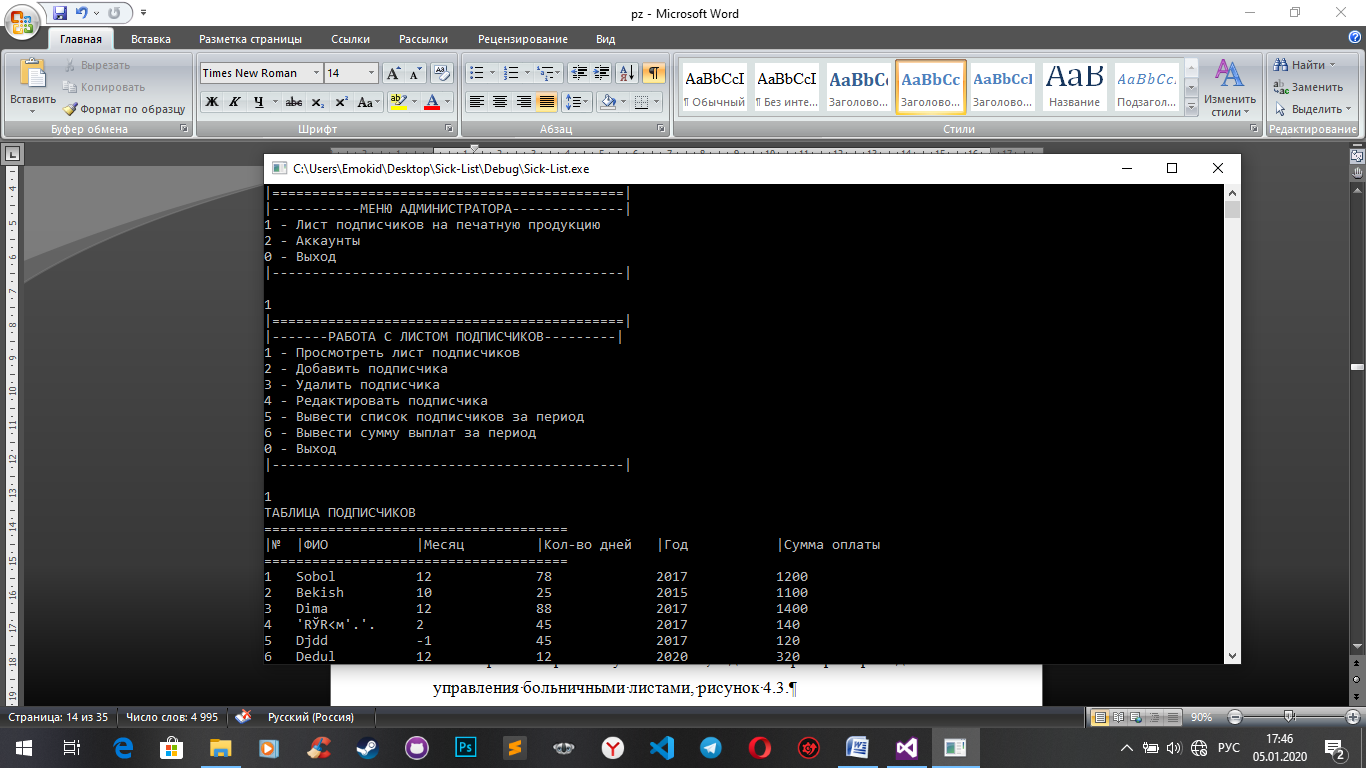


Рисунок 4.2 – Меню администратора

При выборе 1 пункта меню, администратор переходит в меню управления листом подписчиков, рисунок 4.3.

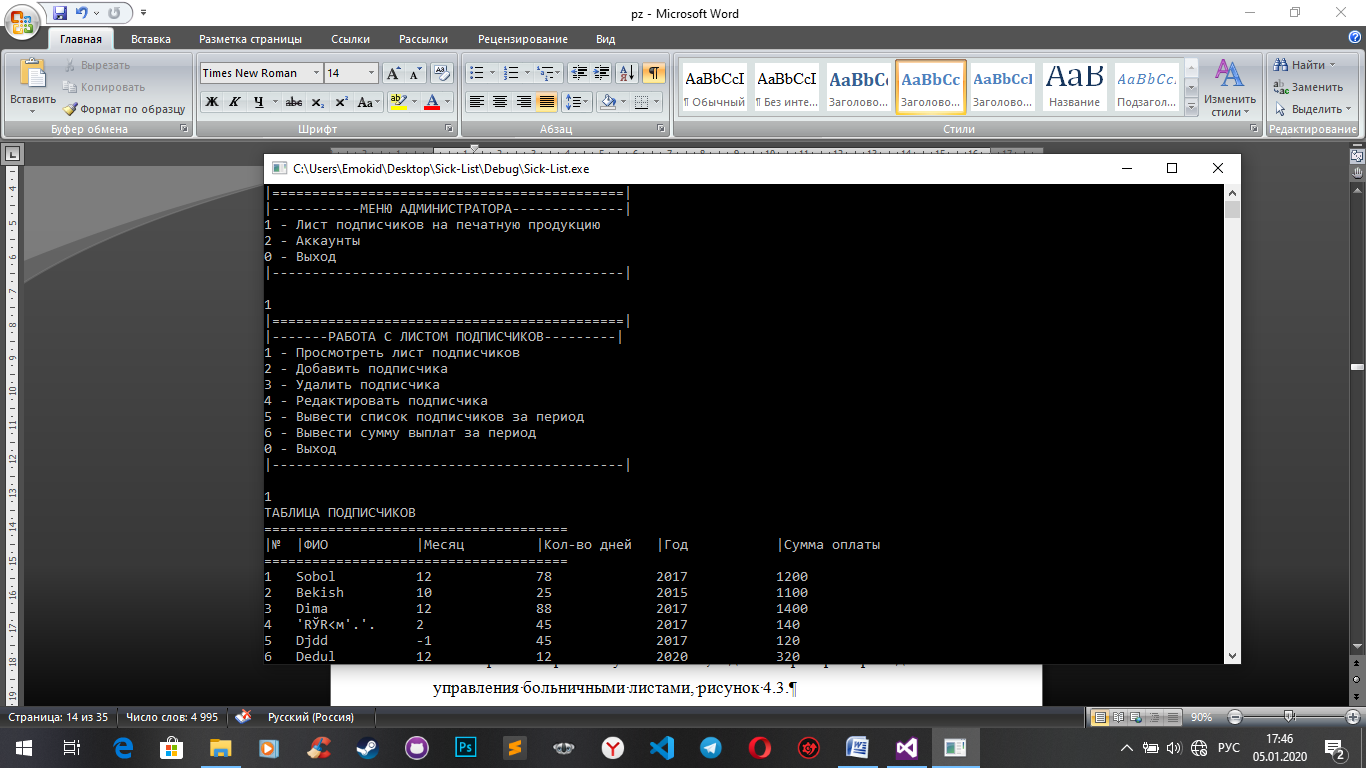


Рисунок 4.3 – Меню управления листом подписчиков

В меню управления листом подписчиков, администратор может просмотреть всех подписчиков, рисунок 4.4.

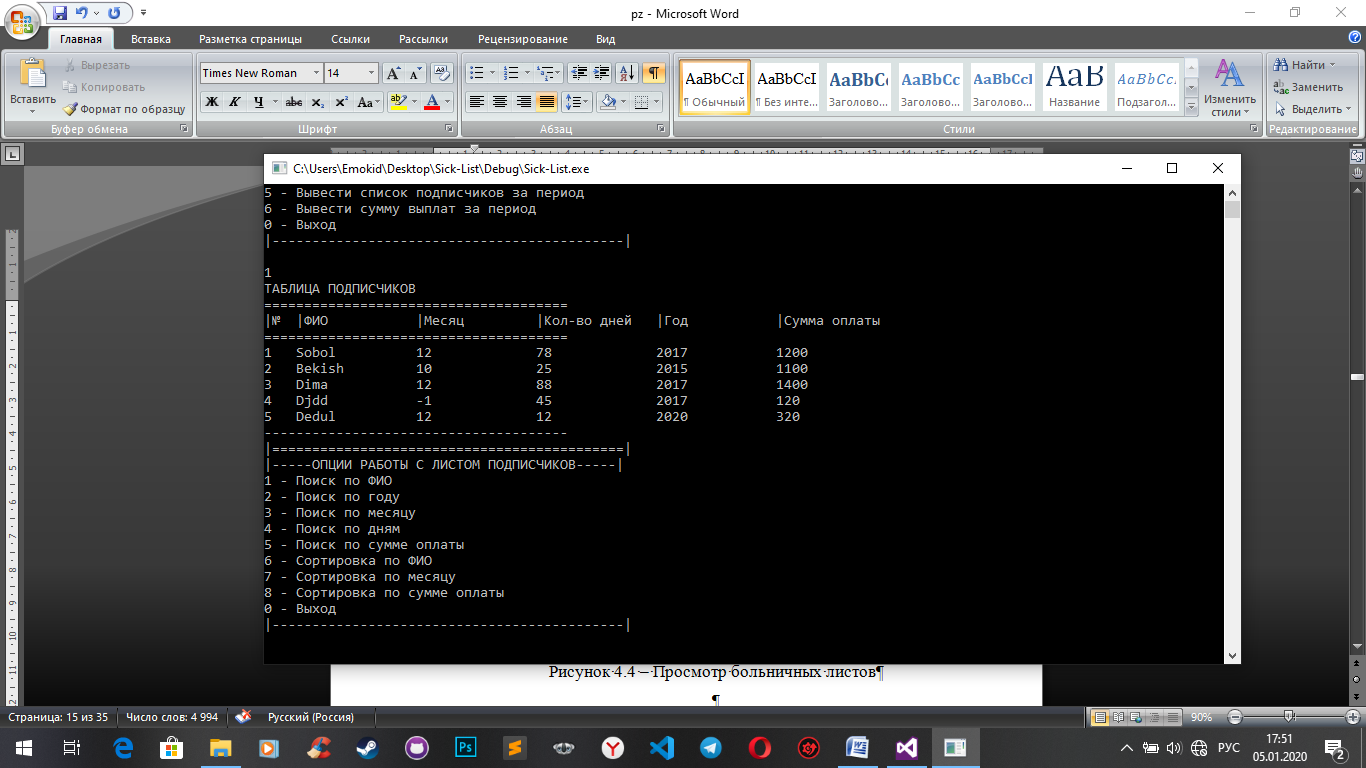


Рисунок 4.4 – Просмотр подписчиков

Также администратор может изменять подписчиков, добавить нового подписчика или удалить подписчика. На рисунке 4.5 отображено добавление нового подписчика.

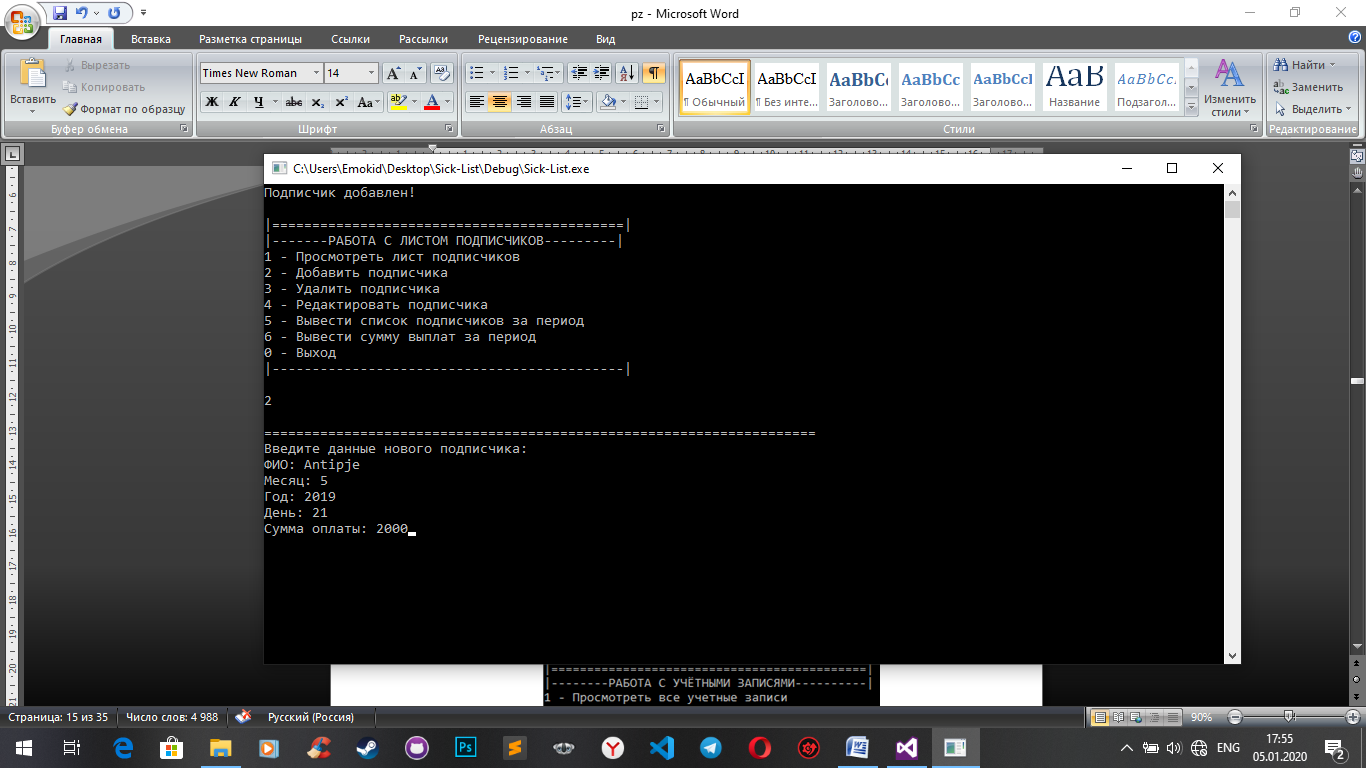


Рисунок 4.5 – Добавление подписчика

При выборе 2 пункта меню, администратор переходит в меню работы с данными об учётных записях рисунок 4.6.

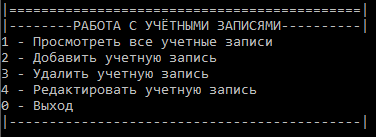


Рисунок 4.6 – Меню учётных записей

В меню учётных записей администратор может добавить новую, отредактировать уже существующую или удалить учётную записью. Также он может просмотреть все существующие учётные записи. На рисунке 4.7 отображено меню редактирования учётной записи.

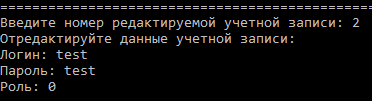


Рисунок 4.7 – Меню редактирования учётной записи

Удаление учётных записей и подписчиков происходит по индексу. На рисунке 4.8, продемонстрировано удаление подписчика.

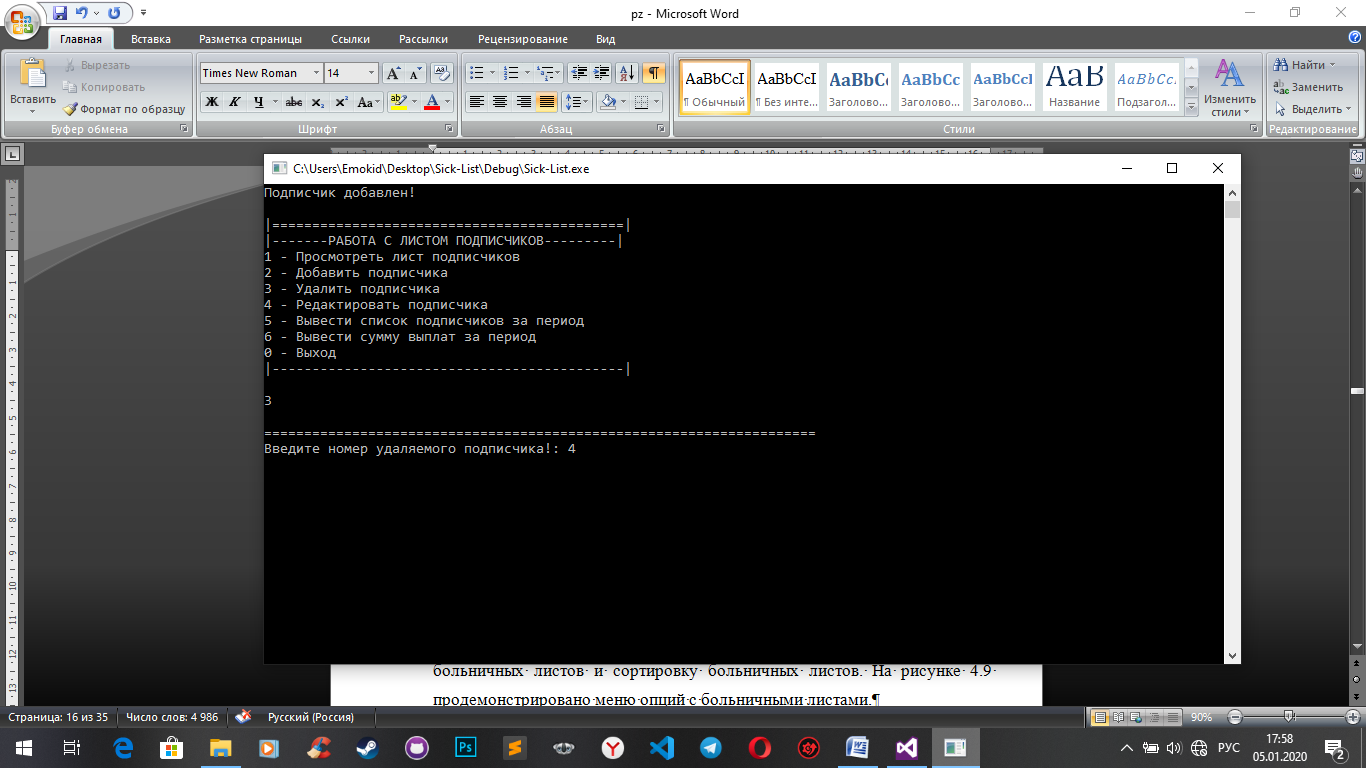




Рисунок 4.8 – Удаление подписчика

В меню опций с листами подписчиков можно производить поиск подписчиков и их сортировку. На рисунке 4.9 продемонстрировано меню опций с листом подписчиков.

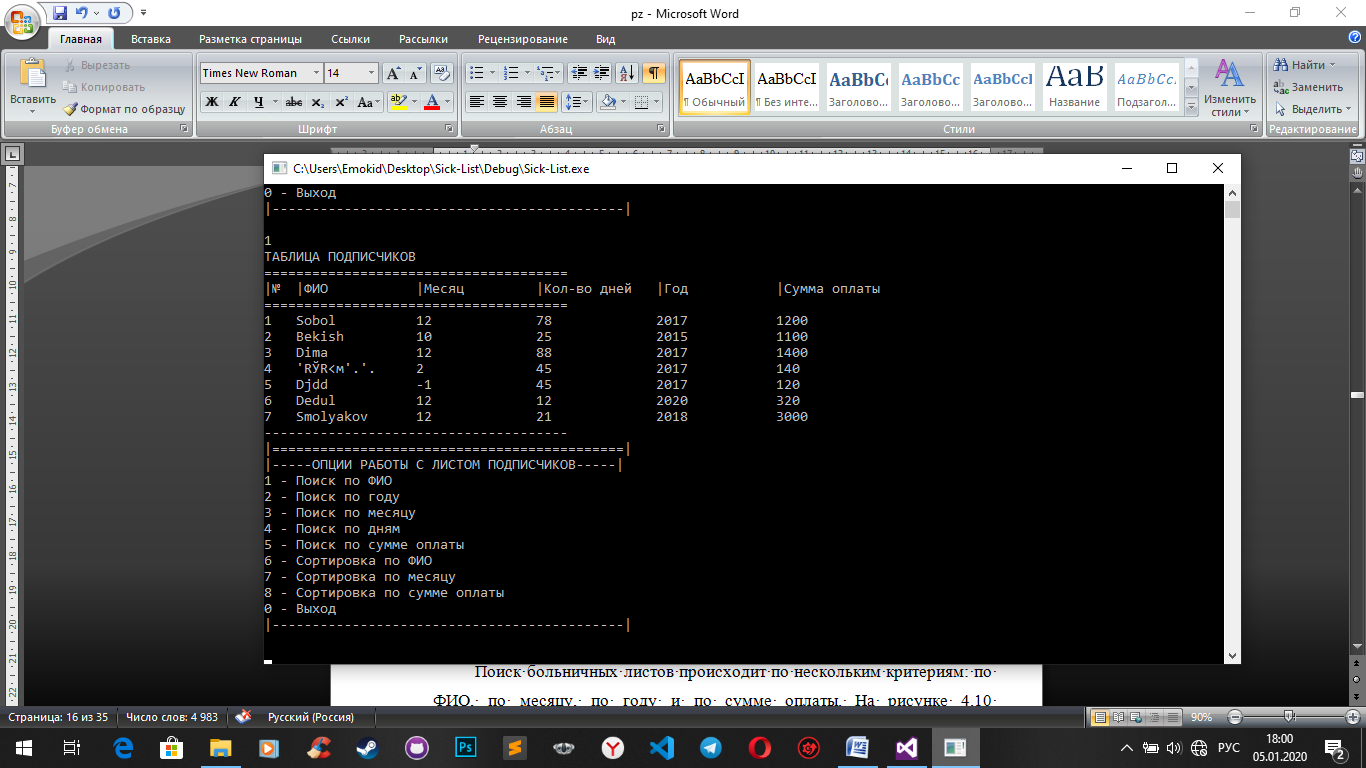


Рисунок 4.9 – Меню опций с листами подписчиков

Поиск подписчиков происходит по нескольким критериям: по ФИО, по месяцу, по году и по сумме оплаты. На рисунке 4.10 продемонстрирован поиск подписчиков по ФИО.

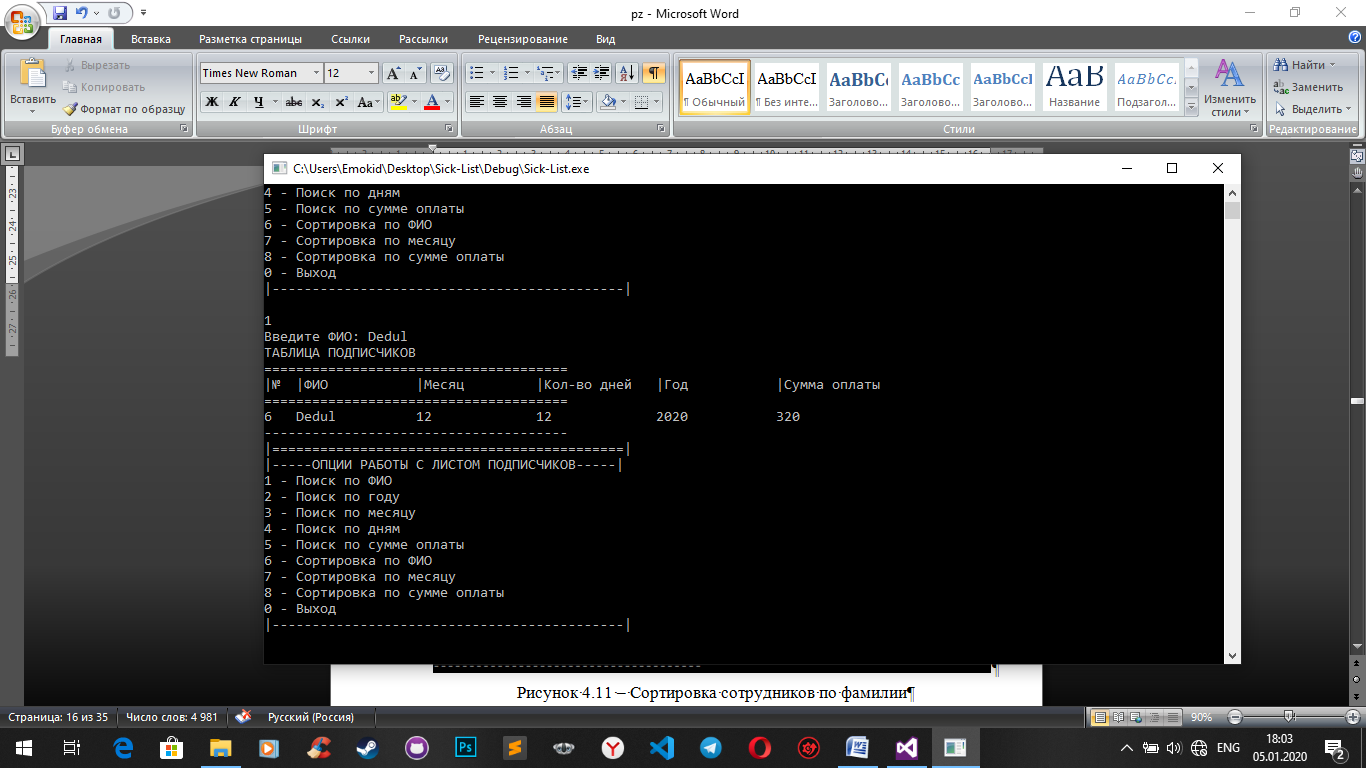


Рисунок 4.10 – Поиск подписчиков по ФИО

Сортировка подписчиков происходит также по нескольким критериям: по ФИО, по месяцу и по сумме оплаты. На рисунке 4.11 продемонстрирована сортировка сотрудников по ФИО.

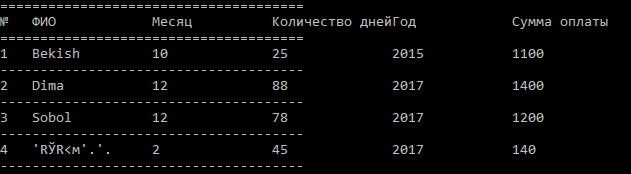


Рисунок 4.11 – Сортировка подписчиков по фамилии

И последним пунктом меню, является выполнение индивидуального задания. В первой части индивидуального задания производится вывод списка подписчиков за определённый период на экран. На рисунке 4.12 продемонстрировано выполнение первой части индивидуального задания.

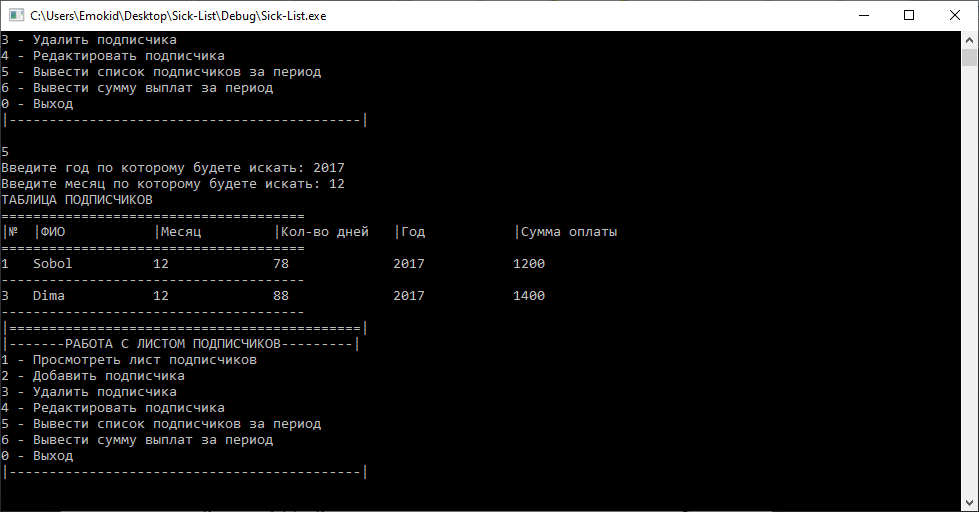


Рисунок 4.14 – Результат поиска подписчиков по заданному периоду

Во второй части индивидуального задания происходит подсчет общей суммы выплат по подписчикам, за месяц. На рисунке 4.13 продемонстрировано выполнение второй части индивидуального задания.



Рисунок 4.13 – Результат суммы по заданному периоду

## 4.3 Модуль пользователя

После авторизации под профилем пользователя мы попадаем в меню пользователя, рисунок 4.15.

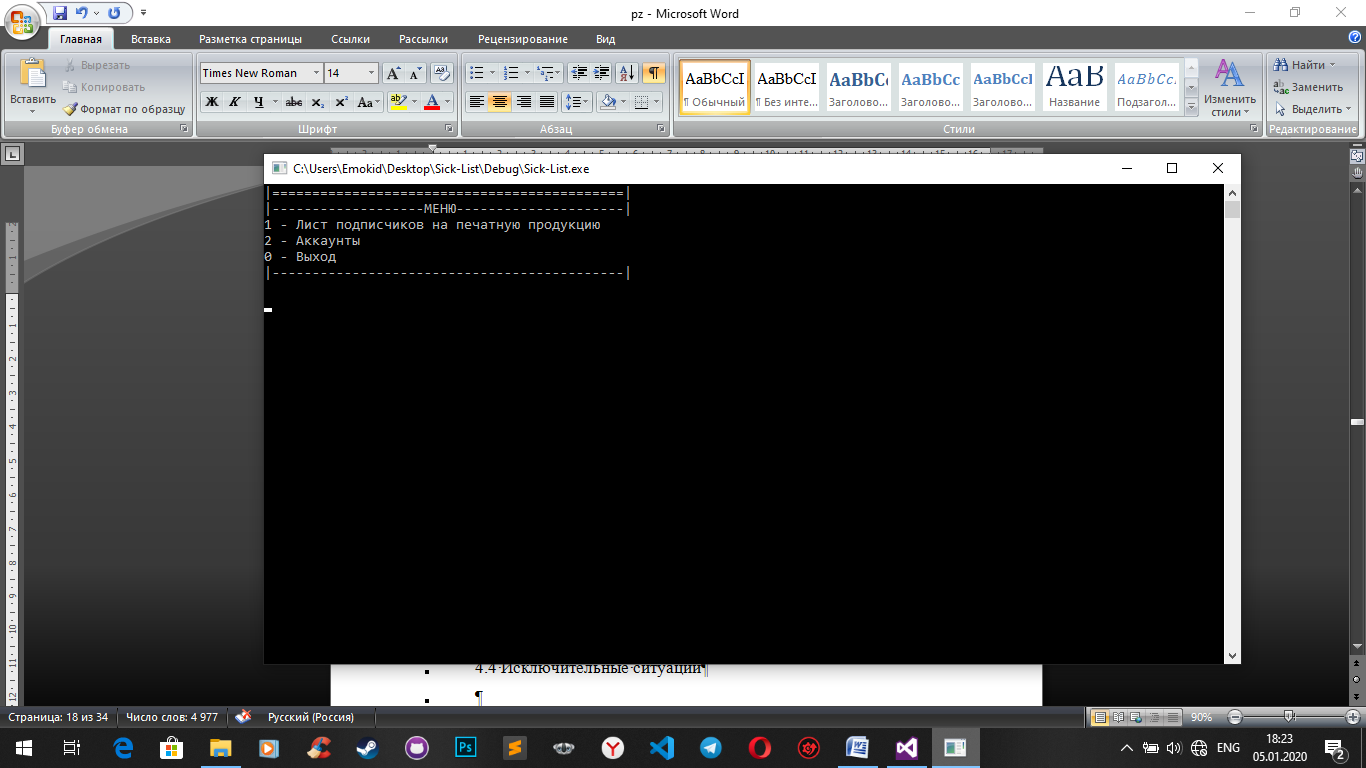


Рисунок 4.15 – Меню пользователя

Сотрудник с ролью «Пользователь» имеет такие же функции, как и администратор, кроме функций связанных с управление учетных записей и функций добавления, изменения и удаления записей о подписчиках.

## 4.4 Исключительные ситуации

Предусмотрено оповещение при входе в программу, рисунок 4.16.



Рисунок 4.16 – Оповещение при входе в программу

При вводе неверного пароля от аккаунта, на экран, окно авторизации отобразится заново, рисунок 4.17.

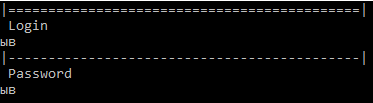


Рисунок 4.17 – Ввод не верного пароля

Заключение

Итогом данного курсового проектирования стала разработка программы учета подписчиков на печатную продукцию, предназначенная для автоматизации учета подписчиков и для облегчения работы персонала.

В ходе разработки программного средства был выполнен анализ предметной области и выявлены требования к программному продукту, на основании чего был сформирован документ «Техническое задание». На этапе проектирования была выполнена реализация функциональных частей и разбиение программы на модули.

Разработанное программное средство полностью соответствует требованиям технического задания.

# Приложение

(обязательное)

Код программы

#include "stdafx.h"

#include <iostream>

#include <iomanip>

#include <string>

#include <vector>

#include <fstream>

#include <cstdlib>

using namespace std;

struct Account

{

string login;

string password;

int role; // 0-пользователь, 1-администратор

};

struct Lists

{

string FIO;

int mounth;

int year;

int days;

int pay;

};

void menu();

void menuAdmin();

void menuAccounts();

void menuList();

void menuListbyAdmin();

void useMenu();

void useMenuAdmin();

void useMenuAccounts();

void useMenuList();

void useMenuTwo();

void useMenuListbyAdmin();

void optionList();

void useMenuOptionList();

void useMenuOptionListbyAdmin();

void menuAutorization(Account \*arr\_of\_accounts, int &number\_of\_accounts);

void writeFileAccounts(Account \*arr\_of\_accounts, int number\_of\_accounts);

void writeEndFileAccounts(Account new\_account);

void readFileAccounts(Account \*arr\_of\_accounts, int &number\_of\_accounts);

void readFileLists(Lists \*arr\_of\_lists, int &number\_of\_lists);

void writeEndFileLists(Lists new\_list);

void writeFileLists(Lists \*arr\_of\_lists, int number\_of\_lists);

void showAccounts(Account \*arr\_of\_accounts, int number\_of\_accounts);

void addAccount(Account \*arr\_of\_accounts, int &number\_of\_accounts);

void delAccount(Account \*arr\_of\_accounts, int &number\_of\_accounts);

void updateAccount(Account \*arr\_of\_accounts, int number\_of\_accounts);

void showLists(Lists \*arr\_of\_lists, int number\_of\_lists);

void showFoundLists(Lists \*arr\_of\_lists, int number\_of\_lists);

void addLists(Lists \*arr\_of\_lists, int &number\_of\_lists);

void delLists(Lists \*arr\_of\_lists, int &number\_of\_lists);

void updateLists(Lists \*arr\_of\_lists, int number\_of\_lists);

void foundList(Lists \*arr\_of\_lists, int number\_of\_lists);

void foundPay(Lists \*arr\_of\_lists, int number\_of\_lists);

void foundForFIO(Lists \*arr\_of\_lists, int number\_of\_lists);

void foundForMounth(Lists \*arr\_of\_lists, int number\_of\_lists);

void foundForYear(Lists \*arr\_of\_lists, int number\_of\_lists);

void foundForDay(Lists \*arr\_of\_lists, int number\_of\_lists);

void foundForPay(Lists \*arr\_of\_lists, int number\_of\_lists);

void SortByFIO(Lists \*arr\_of\_lists, int number\_of\_lists);

void SortByMounth(Lists \*arr\_of\_lists, int number\_of\_lists);

void SortByPay(Lists \*arr\_of\_lists, int number\_of\_lists);

const int SIZE\_ARR\_OF\_\_ACCOUNTS = 100;

string const FILE\_OF\_ACCOUNTS = "Accounts.txt"; //Путь к файлу

const int SIZE\_ARR\_OF\_\_LISTS = 100;

string const FILE\_OF\_LISTS = "Lists.txt";

void main()

{

setlocale(LC\_ALL, "rus");

useMenu();

system("pause");

}

void useMenu()

{

Account arr\_of\_accounts[SIZE\_ARR\_OF\_\_ACCOUNTS];

int number\_of\_accounts = 0;

readFileAccounts(arr\_of\_accounts, number\_of\_accounts);

menuAutorization(arr\_of\_accounts, number\_of\_accounts);

bool flag\_exit = true;

while (flag\_exit)

{

menu();

int item;

cin >> item;

switch (item)

{

case 0: useMenu();

break;

case 1: useMenuList();

break;

case 2: useMenuAccounts();

break;

}

}

writeFileAccounts(arr\_of\_accounts, number\_of\_accounts);

}

void useMenuTwo()

{

Account arr\_of\_accounts[SIZE\_ARR\_OF\_\_ACCOUNTS];

int number\_of\_accounts = 0;

readFileAccounts(arr\_of\_accounts, number\_of\_accounts);

bool flag\_exit = true;

while (flag\_exit)

{

menu();

int item;

cin >> item;

switch (item)

{

case 0: useMenu();

break;

case 1: useMenuList();

break;

case 2: useMenuAccounts();

break;

}

}

writeFileAccounts(arr\_of\_accounts, number\_of\_accounts);

}

void useMenuAdmin()

{

Account arr\_of\_accounts[SIZE\_ARR\_OF\_\_ACCOUNTS];

int number\_of\_accounts = 0;

readFileAccounts(arr\_of\_accounts, number\_of\_accounts);

bool flag\_exit = true;

while (flag\_exit)

{

menuAdmin();

int item;

cin >> item;

switch (item)

{

case 0: useMenu();

break;

case 1: useMenuListbyAdmin();

break;

case 2: useMenuAccounts();

break;

}

}

writeFileAccounts(arr\_of\_accounts, number\_of\_accounts);

}

void useMenuAccounts()

{

Account arr\_of\_accounts[SIZE\_ARR\_OF\_\_ACCOUNTS];

int number\_of\_accounts = 0;

readFileAccounts(arr\_of\_accounts, number\_of\_accounts);

bool flag\_exit = true;

while (flag\_exit)

{

menuAccounts();

int item;

cin >> item;

switch (item)

{

case 0: flag\_exit = false;

break;

case 1: showAccounts(arr\_of\_accounts, number\_of\_accounts);

break;

case 2: addAccount(arr\_of\_accounts, number\_of\_accounts);

break;

case 3: delAccount(arr\_of\_accounts, number\_of\_accounts);

break;

case 4: updateAccount(arr\_of\_accounts, number\_of\_accounts);

break;

}

}

writeFileAccounts(arr\_of\_accounts, number\_of\_accounts);

}

void useMenuOptionList()

{

Lists arr\_of\_lists[SIZE\_ARR\_OF\_\_LISTS];

int number\_of\_lists = 0;

readFileLists(arr\_of\_lists, number\_of\_lists);

bool flag\_exit = true;

while (flag\_exit)

{

optionList();

int item;

cin >> item;

switch (item)

{

case 0: useMenuList();

break;

case 1: foundForFIO(arr\_of\_lists, number\_of\_lists);

break;

case 2: foundForYear(arr\_of\_lists, number\_of\_lists);

break;

case 3: foundForMounth(arr\_of\_lists, number\_of\_lists);

break;

case 4: foundForDay(arr\_of\_lists, number\_of\_lists);

break;

case 5: foundForPay(arr\_of\_lists, number\_of\_lists);

break;

case 6: SortByFIO(arr\_of\_lists, number\_of\_lists);

break;

case 7: SortByMounth(arr\_of\_lists, number\_of\_lists);

break;

case 8: SortByPay(arr\_of\_lists, number\_of\_lists);

break;

}

}

writeFileLists(arr\_of\_lists, number\_of\_lists);

}

void useMenuOptionListbyAdmin()

{

Lists arr\_of\_lists[SIZE\_ARR\_OF\_\_LISTS];

int number\_of\_lists = 0;

readFileLists(arr\_of\_lists, number\_of\_lists);

bool flag\_exit = true;

while (flag\_exit)

{

optionList();

int item;

cin >> item;

switch (item)

{

case 0: useMenuListbyAdmin();

break;

case 1: foundForFIO(arr\_of\_lists, number\_of\_lists);

break;

case 2: foundForYear(arr\_of\_lists, number\_of\_lists);

break;

case 3: foundForMounth(arr\_of\_lists, number\_of\_lists);

break;

case 4: foundForDay(arr\_of\_lists, number\_of\_lists);

break;

case 5: foundForPay(arr\_of\_lists, number\_of\_lists);

break;

case 6: SortByFIO(arr\_of\_lists, number\_of\_lists);

break;

case 7: SortByMounth(arr\_of\_lists, number\_of\_lists);

break;

case 8: SortByPay(arr\_of\_lists, number\_of\_lists);

break;

}

}

writeFileLists(arr\_of\_lists, number\_of\_lists);

}

void useMenuList()

{

Lists arr\_of\_lists[SIZE\_ARR\_OF\_\_LISTS];

int number\_of\_lists = 0;

readFileLists(arr\_of\_lists, number\_of\_lists);

bool flag\_exit = true;

while (flag\_exit)

{

menuList();

int item;

cin >> item;

switch (item)

{

case 0: useMenuTwo();

break;

case 1: showLists(arr\_of\_lists, number\_of\_lists);

useMenuOptionList();

break;

case 2: foundList(arr\_of\_lists, number\_of\_lists);

break;

case 3: foundPay(arr\_of\_lists, number\_of\_lists);

break;

}

}

writeFileLists(arr\_of\_lists, number\_of\_lists);

}

void useMenuListbyAdmin()

{

Lists arr\_of\_lists[SIZE\_ARR\_OF\_\_LISTS];

int number\_of\_lists = 0;

readFileLists(arr\_of\_lists, number\_of\_lists);

bool flag\_exit = true;

while (flag\_exit)

{

menuListbyAdmin();

int item;

cin >> item;

switch (item)

{

case 0: useMenuAdmin();

break;

case 1: showLists(arr\_of\_lists, number\_of\_lists);

useMenuOptionListbyAdmin();

break;

case 2: addLists(arr\_of\_lists, number\_of\_lists);

break;

case 3: delLists(arr\_of\_lists, number\_of\_lists);

break;

case 4: updateLists(arr\_of\_lists, number\_of\_lists);

break;

case 5: foundList(arr\_of\_lists, number\_of\_lists);

break;

case 6: foundPay(arr\_of\_lists, number\_of\_lists);

break;

}

}

writeFileLists(arr\_of\_lists, number\_of\_lists);

}

void menuAutorization(Account \*arr\_of\_accounts, int &number\_of\_accounts)

{

bool flag = true;

do

{

system("cls");

string alogin;

string apassword;

cout << "|============================================|" << endl;

cout << " Login" << endl;

cin >> alogin;

cout << "|--------------------------------------------|" << endl;

cout << " Password" << endl;

cin >> apassword;

cout << "|============================================|" << endl;

ifstream fin(FILE\_OF\_ACCOUNTS, ios::in); //Открыли файл для чтения

if (fin.is\_open())

{

for (int i = 0; i < number\_of\_accounts; i++)

{

if ((arr\_of\_accounts[i].login == alogin) && (arr\_of\_accounts[i].password == apassword))

{

system("cls");

if (arr\_of\_accounts[i].role == 1)

{

cout << "Вы успешно авторизировались!" << endl;

system("pause");

system("cls");

useMenuAdmin();

}

else

{

flag = false;

}

}

}

}

} while (flag == true);

cout << endl;

cout << "Вы успешно авторизировались!" << endl;

system("pause");

system("cls");

}

void menuAdmin()

{

cout << "|============================================|" << endl;

cout << "|-----------МЕНЮ АДМИНИСТРАТОРА--------------|" << endl;

cout << "1 - Лист подписчиков на печатную продукцию" << endl;

cout << "2 - Аккаунты " << endl;

cout << "0 - Выход" << endl;

cout << "|--------------------------------------------|" << endl;

cout << endl;

}

void menu()

{

cout << "|============================================|" << endl;

cout << "|-------------------МЕНЮ---------------------|" << endl;

cout << "1 - Лист подписчиков на печатную продукцию" << endl;

cout << "2 - Аккаунты " << endl;

cout << "0 - Выход" << endl;

cout << "|--------------------------------------------|" << endl;

cout << endl;

}

void menuList()

{

cout << "|============================================|" << endl;

cout << "|-------РАБОТА С БОЛЬНИЧНЫМИ ЛИСТАМИ---------|" << endl;

cout << "1 - Просмотреть лист подписчиков" << endl;

cout << "2 - Вывести список подписчиков за период" << endl;

cout << "3 - Вывести сумму выплат за период" << endl;

cout << "0 - Выход" << endl;

cout << "|--------------------------------------------|" << endl;

cout << endl;

}

void menuListbyAdmin()

{

cout << "|============================================|" << endl;

cout << "|-------РАБОТА С ЛИСТОМ ПОДПИСЧИКОВ---------|" << endl;

cout << "1 - Просмотреть лист подписчиков" << endl;

cout << "2 - Добавить подписчика" << endl;

cout << "3 - Удалить подписчика" << endl;

cout << "4 - Редактировать подписчика" << endl;

cout << "5 - Вывести список подписчиков за период" << endl;

cout << "6 - Вывести сумму выплат за период" << endl;

cout << "0 - Выход" << endl;

cout << "|--------------------------------------------|" << endl;

cout << endl;

}

void menuAccounts()

{

cout << "|============================================|" << endl;

cout << "|--------РАБОТА С УЧЁТНЫМИ ЗАПИСЯМИ----------|" << endl;

cout << "1 - Просмотреть все учетные записи" << endl;

cout << "2 - Добавить учетную запись" << endl;

cout << "3 - Удалить учетную запись" << endl;

cout << "4 - Редактировать учетную запись" << endl;

cout << "0 - Выход" << endl;

cout << "|--------------------------------------------|" << endl;

cout << endl;

}

void optionList()

{

cout << "|============================================|" << endl;

cout << "|-----ОПЦИИ РАБОТЫ С ЛИСТОМ ПОДПИСЧИКОВ-----|" << endl;

cout << "1 - Поиск по ФИО" << endl;

cout << "2 - Поиск по году" << endl;

cout << "3 - Поиск по месяцу" << endl;

cout << "4 - Поиск по дням" << endl;

cout << "5 - Поиск по сумме оплаты" << endl;

cout << "6 - Сортировка по ФИО" << endl;

cout << "7 - Сортировка по месяцу" << endl;

cout << "8 - Сортировка по сумме оплаты" << endl;

cout << "0 - Выход" << endl;

cout << "|--------------------------------------------|" << endl;

cout << endl;

}

void readFileAccounts(Account \*arr\_of\_accounts, int &number\_of\_accounts)

{

ifstream fin(FILE\_OF\_ACCOUNTS, ios::in); //Открыли файл для чтения

if (!fin.is\_open())

{

arr\_of\_accounts[0].login = "admin";

arr\_of\_accounts[0].password = "admin";

arr\_of\_accounts[0].role = 1;

number\_of\_accounts = 1;

writeEndFileAccounts(arr\_of\_accounts[0]);

}

else

{

number\_of\_accounts = 0;

while (!fin.eof())

{

fin >> arr\_of\_accounts[number\_of\_accounts].login >> arr\_of\_accounts[number\_of\_accounts].password >> arr\_of\_accounts[number\_of\_accounts].role;

number\_of\_accounts++;

}

}

fin.close(); //Закрыли открытый файл

}

void readFileLists(Lists \*arr\_of\_lists, int &number\_of\_lists)

{

ifstream fin(FILE\_OF\_LISTS, ios::in); //Открыли файл для чтения

number\_of\_lists = 0;

while (!fin.eof())

{

fin >> arr\_of\_lists[number\_of\_lists].FIO >> arr\_of\_lists[number\_of\_lists].mounth >> arr\_of\_lists[number\_of\_lists].year >> arr\_of\_lists[number\_of\_lists].days >> arr\_of\_lists[number\_of\_lists].pay;

number\_of\_lists++;

}

fin.close(); //Закрыли открытый файл

}

void writeFileAccounts(Account \*arr\_of\_accounts, int number\_of\_accounts)

{

ofstream fout(FILE\_OF\_ACCOUNTS, ios::out); //Открыли файл для записи

for (int i = 0; i < number\_of\_accounts; i++)

{

if (i < number\_of\_accounts - 1)

fout << arr\_of\_accounts[i].login << " " << arr\_of\_accounts[i].password << " " << arr\_of\_accounts[i].role << endl;

else

fout << arr\_of\_accounts[i].login << " " << arr\_of\_accounts[i].password << " " << arr\_of\_accounts[i].role;

}

fout.close();

}

void writeFileLists(Lists \*arr\_of\_lists, int number\_of\_lists)

{

ofstream fout(FILE\_OF\_LISTS, ios::out); //Открыли файл для записи

for (int i = 0; i < number\_of\_lists; i++)

{

if (i < number\_of\_lists - 1)

fout << arr\_of\_lists[i].FIO << " " << arr\_of\_lists[i].mounth << " " << arr\_of\_lists[i].year << " " << arr\_of\_lists[i].days << " " << arr\_of\_lists[i].pay << endl;

else

fout << arr\_of\_lists[i].FIO << " " << arr\_of\_lists[i].mounth << " " << arr\_of\_lists[i].year << " " << arr\_of\_lists[i].days << " " << arr\_of\_lists[i].pay;

}

fout.close();

}

void writeEndFileAccounts(Account new\_account)

{

ofstream fout(FILE\_OF\_ACCOUNTS, ios::app); //Открыли файл для ДОзаписи

fout << endl;

fout << new\_account.login << " " << new\_account.password << " " << new\_account.role;

fout.close();

}

void writeEndFileLists(Lists new\_lists)

{

ofstream fout(FILE\_OF\_LISTS, ios::app); //Открыли файл для ДОзаписи

fout << endl;

fout << new\_lists.FIO << " " << new\_lists.mounth << " " << new\_lists.year << " " << new\_lists.days << " " << new\_lists.pay;

fout.close();

}

void showAccounts(Account \*arr\_of\_accounts, int number\_of\_accounts)

{

cout << "ТАБЛИЦА УЧЕТНЫХ ЗАПИСЕЙ ПОЛЬЗОВАТЕЛЕЙ" << endl;

cout << setw(37) << "======================================" << endl;

cout << setw(4) << left << "№ " << setw(15) << left << "Логин" << setw(15) << left << "Пароль" << setw(3) << left << "Роль" << endl;

cout << setw(37) << "======================================" << endl;

for (int i = 0; i < number\_of\_accounts; i++)

cout << setw(4) << left << i + 1 <<

setw(15) << left << arr\_of\_accounts[i].login <<

setw(15) << left << arr\_of\_accounts[i].password <<

setw(3) << left << arr\_of\_accounts[i].role << endl;

cout << setw(37) << "--------------------------------------" << endl;

}

void showLists(Lists \*arr\_of\_lists, int number\_of\_lists)

{

cout << "ТАБЛИЦА ПОДПИСЧИКОВ" << endl;

cout << setw(37) << "======================================" << endl;

cout << setw(4) << left << "|№ " << setw(15) << left << "|ФИО" << setw(15) << left << "|Месяц" << setw(15) << left << "|Кол-во дней" << setw(15) << left << "|Год" << setw(15) << left << "|Сумма оплаты" << endl;

cout << setw(37) << "======================================" << endl;

for (int i = 0; i < number\_of\_lists; i++)

cout << setw(4) << left << i + 1 <<

setw(15) << left << arr\_of\_lists[i].FIO <<

setw(15) << left << arr\_of\_lists[i].mounth <<

setw(15) << left << arr\_of\_lists[i].days <<

setw(15) << left << arr\_of\_lists[i].year <<

setw(15) << left << arr\_of\_lists[i].pay << endl;

cout << setw(37) << "--------------------------------------" << endl;

}

void showFoundLists(Lists \*arr\_of\_lists, int number\_of\_lists)

{

cout << setw(4) << left << number\_of\_lists + 1 <<

setw(15) << left << arr\_of\_lists[number\_of\_lists].FIO <<

setw(15) << left << arr\_of\_lists[number\_of\_lists].mounth <<

setw(15) << left << arr\_of\_lists[number\_of\_lists].days <<

setw(15) << left << arr\_of\_lists[number\_of\_lists].year <<

setw(15) << left << arr\_of\_lists[number\_of\_lists].pay << endl;

cout << setw(37) << "--------------------------------------" << endl;

}

void addAccount(Account \*arr\_of\_accounts, int &number\_of\_accounts)

{

number\_of\_accounts++;

cout << endl;

cout << "=====================================================================" << endl;

cout << "Введите данные новой учетной записи:" << endl;

cout << "Логин: ";

cin >> arr\_of\_accounts[number\_of\_accounts - 1].login;

cout << "Пароль: ";

cin >> arr\_of\_accounts[number\_of\_accounts - 1].password;

cout << "Роль: ";

cin >> arr\_of\_accounts[number\_of\_accounts - 1].role;

if ((arr\_of\_accounts[number\_of\_accounts - 1].role == 0) || (arr\_of\_accounts[number\_of\_accounts - 1].role == 1))

{

writeEndFileAccounts(arr\_of\_accounts[number\_of\_accounts - 1]);

system("cls");

cout << "Новая учетная запись добавлена!" << endl;

cout << endl;

}

else

{

arr\_of\_accounts[number\_of\_accounts - 1].login = "";

arr\_of\_accounts[number\_of\_accounts - 1].password = "";

arr\_of\_accounts[number\_of\_accounts - 1].role = NULL;

cout << "Неправильно выбрана роль!" << endl;

}

}

void addLists(Lists \*arr\_of\_lists, int &number\_of\_lists)

{

int i;

bool FLAG = true;

number\_of\_lists++;

cout << endl;

cout << "=====================================================================" << endl;

cout << "Введите данные нового подписчика:" << endl;

cout << "ФИО: ";

cin >> arr\_of\_lists[number\_of\_lists - 1].FIO;

cout << "Месяц: ";

cin >> arr\_of\_lists[number\_of\_lists - 1].mounth;

cout << "Год: ";

cin >> arr\_of\_lists[number\_of\_lists - 1].year;

cout << "День: ";

cin >> arr\_of\_lists[number\_of\_lists - 1].days;

cout << "Сумма оплаты: ";

cin >> arr\_of\_lists[number\_of\_lists - 1].pay;

writeEndFileLists(arr\_of\_lists[number\_of\_lists - 1]);

system("cls");

cout << "Подписчик добавлен!" << endl;

cout << endl;

}

void delAccount(Account \*arr\_of\_accounts, int &number\_of\_accounts)

{

cout << endl;

cout << "=====================================================================" << endl;

cout << "Введите номер удаляемой учетной записи: ";

int del\_item;

cin >> del\_item;

for (int i = del\_item - 1; i < number\_of\_accounts - 1; i++)

arr\_of\_accounts[i] = arr\_of\_accounts[i + 1];

number\_of\_accounts--;

system("cls");

cout << "Учетная запись удалена!" << endl;

cout << endl;

}

void delLists(Lists \*arr\_of\_lists, int &number\_of\_lists)

{

cout << endl;

cout << "=====================================================================" << endl;

cout << "Введите номер удаляемого подписчика!: ";

int del\_item;

cin >> del\_item;

for (int i = del\_item - 1; i < number\_of\_lists - 1; i++)

arr\_of\_lists[i] = arr\_of\_lists[i + 1];

number\_of\_lists--;

system("cls");

cout << "Учетная запись удалена!" << endl;

cout << endl;

}

void updateAccount(Account \*arr\_of\_accounts, int number\_of\_accounts)

{

cout << endl;

cout << "=====================================================================" << endl;

cout << "Введите номер редактируемой учетной записи: ";

int update\_item;

cin >> update\_item;

cout << "Отредактируйте данные учетной записи:" << endl;

cout << "Логин: ";

cin >> arr\_of\_accounts[update\_item - 1].login;

cout << "Пароль: ";

cin >> arr\_of\_accounts[update\_item - 1].password;

cout << "Роль: ";

cin >> arr\_of\_accounts[update\_item - 1].role;

system("cls");

cout << "Изменения успешно внесены!" << endl;

cout << endl;

}

void updateLists(Lists \*arr\_of\_lists, int number\_of\_lists)

{

cout << endl;

cout << "=====================================================================" << endl;

cout << "Введите номер редактируемой учетной записи: ";

int update\_item;

cin >> update\_item;

cout << "Отредактируйте данные подписчика:" << endl;

cout << "ФИО: ";

cin >> arr\_of\_lists[number\_of\_lists - 1].FIO;

cout << "Месяц: ";

cin >> arr\_of\_lists[number\_of\_lists - 1].mounth;

cout << "Год: ";

cin >> arr\_of\_lists[number\_of\_lists - 1].days;

cout << "День: ";

cin >> arr\_of\_lists[number\_of\_lists - 1].mounth;

cout << "Сумма оплаты: ";

cin >> arr\_of\_lists[number\_of\_lists - 1].days;

system("cls");

cout << "Изменения успешно внесены!" << endl;

cout << endl;

}

void foundList(Lists \*arr\_of\_lists, int number\_of\_lists)

{

int FoundYear = 0;

int FoundMounth = 0;

cout << "Введите год по которому будете искать: ";

(cin >> FoundYear).get();

cout << "Введите месяц по которому будете искать: ";

(cin >> FoundMounth).get();

if (cin.fail())

{

cout << "Год и месяц должны быть указаны цифрами" << endl;

system("pause");

}

cout << "ТАБЛИЦА ПОДПИСЧИКОВ" << endl;

cout << setw(37) << "======================================" << endl;

cout << setw(4) << left << "|№ " << setw(15) << left << "|ФИО" << setw(15) << left << "|Месяц" << setw(15) << left << "|Кол-во дней" << setw(15) << left << "|Год" << setw(15) << left << "|Сумма оплаты" << endl;

cout << setw(37) << "======================================" << endl;

for (int i = 0; i < number\_of\_lists; i++)

{

if ((arr\_of\_lists[i].mounth == FoundMounth) && (arr\_of\_lists[i].year == FoundYear))

{

showFoundLists(arr\_of\_lists, i);

}

}

}

void foundPay(Lists \*arr\_of\_lists, int number\_of\_lists)

{

int sum = 0;

int FoundYear = 0;

int FoundMounth = 0;

cout << "Введите год по которому будете искать: ";

(cin >> FoundYear).get();

cout << "Введите месяц по которому будете искать: ";

(cin >> FoundMounth).get();

if (cin.fail())

{

cout << "Год и месяц должны быть указаны цифрами" << endl;

system("pause");

}

for (int i = 0; i < number\_of\_lists; i++)

{

if ((arr\_of\_lists[i].mounth == FoundMounth) && (arr\_of\_lists[i].year == FoundYear))

{

sum = sum + arr\_of\_lists[i].pay;

}

}

cout << "Сумма за выбранный период равна - " << sum << endl;

}

void foundForFIO(Lists \*arr\_of\_lists, int number\_of\_lists)

{

string FIO;

cout << "Введите ФИО: ";

(cin >> FIO).get();

cout << "ТАБЛИЦА ПОДПИСЧИКОВ" << endl;

cout << setw(37) << "======================================" << endl;

cout << setw(4) << left << "|№ " << setw(15) << left << "|ФИО" << setw(15) << left << "|Месяц" << setw(15) << left << "|Кол-во дней" << setw(15) << left << "|Год" << setw(15) << left << "|Сумма оплаты" << endl;

cout << setw(37) << "======================================" << endl;

for (int i = 0; i < number\_of\_lists; i++)

{

if (arr\_of\_lists[i].FIO == FIO)

{

showFoundLists(arr\_of\_lists, i);

}

}

}

void foundForMounth(Lists \*arr\_of\_lists, int number\_of\_lists)

{

int mounth;

cout << "Введите месяц: ";

(cin >> mounth).get();

cout << "ТАБЛИЦА ПОДПИСЧИКОВ" << endl;

cout << setw(37) << "======================================" << endl;

cout << setw(4) << left << "|№ " << setw(15) << left << "|ФИО" << setw(15) << left << "|Месяц" << setw(15) << left << "|Кол-во дней" << setw(15) << left << "|Год" << setw(15) << left << "|Сумма оплаты" << endl;

cout << setw(37) << "======================================" << endl;

for (int i = 0; i < number\_of\_lists; i++)

{

if (arr\_of\_lists[i].mounth == mounth)

{

showFoundLists(arr\_of\_lists, i);

}

}

}

void foundForDay(Lists \*arr\_of\_lists, int number\_of\_lists)

{

int days;

cout << "Введите количество дней: ";

(cin >> days).get();

cout << "ТАБЛИЦА ПОДПИСЧИКОВ" << endl;

cout << setw(37) << "======================================" << endl;

cout << setw(4) << left << "|№ " << setw(15) << left << "|ФИО" << setw(15) << left << "|Месяц" << setw(15) << left << "|Кол-во дней" << setw(15) << left << "|Год" << setw(15) << left << "|Сумма оплаты" << endl;

cout << setw(37) << "======================================" << endl;

for (int i = 0; i < number\_of\_lists; i++)

{

if (arr\_of\_lists[i].days == days)

{

showFoundLists(arr\_of\_lists, i);

}

}

}

void foundForYear(Lists \*arr\_of\_lists, int number\_of\_lists)

{

int year;

cout << "Введите год: ";

(cin >> year).get();

cout << "ТАБЛИЦА ПОДПИСЧИКОВ" << endl;

cout << setw(37) << "======================================" << endl;

cout << setw(4) << left << "|№ " << setw(15) << left << "|ФИО" << setw(15) << left << "|Месяц" << setw(15) << left << "|Кол-во дней" << setw(15) << left << "|Год" << setw(15) << left << "|Сумма оплаты" << endl;

cout << setw(37) << "======================================" << endl;

for (int i = 0; i < number\_of\_lists; i++)

{

if (arr\_of\_lists[i].year == year)

{

showFoundLists(arr\_of\_lists, i);

}

}

}

void foundForPay(Lists \*arr\_of\_lists, int number\_of\_lists)

{

int pay;

cout << "Введите сумму выплаты: ";

(cin >> pay).get();

cout << "ТАБЛИЦА ПОДПИСЧИКОВ" << endl;

cout << setw(37) << "======================================" << endl;

cout << setw(4) << left << "|№ " << setw(15) << left << "|ФИО" << setw(15) << left << "|Месяц" << setw(15) << left << "|Кол-во дней" << setw(15) << left << "|Год" << setw(15) << left << "|Сумма оплаты" << endl;

cout << setw(37) << "======================================" << endl;

for (int i = 0; i < number\_of\_lists; i++)

{

if (arr\_of\_lists[i].pay == pay)

{

showFoundLists(arr\_of\_lists, i);

}

}

}

void SortByFIO(Lists \*arr\_of\_lists, int number\_of\_lists)

{

Lists tmp;

system("cls");

cout << setw(37) << "======================================" << endl;

cout << setw(4) << left << "|№ " << setw(15) << left << "|ФИО" << setw(15) << left << "|Месяц" << setw(15) << left << "|Кол-во дней" << setw(15) << left << "|Год" << setw(15) << left << "|Сумма оплаты" << endl;

cout << setw(37) << "======================================" << endl;

for (int i = 0; i < number\_of\_lists; i++)

{

for (int j = 0; j < number\_of\_lists - 1; j++)

{

if (arr\_of\_lists[j].FIO > arr\_of\_lists[j + 1].FIO)

{

tmp = arr\_of\_lists[j + 1];

arr\_of\_lists[j + 1] = arr\_of\_lists[j];

arr\_of\_lists[j] = tmp;

}

}

showFoundLists(arr\_of\_lists, i);

}

}

void SortByMounth(Lists \*arr\_of\_lists, int number\_of\_lists)

{

Lists tmp;

system("cls");

cout << setw(37) << "======================================" << endl;

cout << setw(4) << left << "|№ " << setw(15) << left << "|ФИО" << setw(15) << left << "|Месяц" << setw(15) << left << "|Кол-во дней" << setw(15) << left << "|Год" << setw(15) << left << "|Сумма оплаты" << endl;

cout << setw(37) << "======================================" << endl;

for (int i = 0; i < number\_of\_lists; i++)

{

for (int j = 0; j < number\_of\_lists - 1; j++)

{

if (arr\_of\_lists[j].mounth > arr\_of\_lists[j + 1].mounth)

{

tmp = arr\_of\_lists[j + 1];

arr\_of\_lists[j + 1] = arr\_of\_lists[j];

arr\_of\_lists[j] = tmp;

}

}

showFoundLists(arr\_of\_lists, i);

}

}

void SortByPay(Lists \*arr\_of\_lists, int number\_of\_lists)

{

Lists tmp;

system("cls");

cout << setw(37) << "======================================" << endl;

cout << setw(4) << left << "|№ " << setw(15) << left << "|ФИО" << setw(15) << left << "|Месяц" << setw(15) << left << "|Кол-во дней" << setw(15) << left << "|Год" << setw(15) << left << "|Сумма оплаты" << endl;

cout << setw(37) << "======================================" << endl;

for (int i = 0; i < number\_of\_lists; i++)

{

for (int j = 0; j < number\_of\_lists - 1; j++)

{

if (arr\_of\_lists[j].pay > arr\_of\_lists[j + 1].pay)

{

tmp = arr\_of\_lists[j + 1];

arr\_of\_lists[j + 1] = arr\_of\_lists[j];

arr\_of\_lists[j] = tmp;

}

}

showFoundLists(arr\_of\_lists, i);

}

}