Содержание

[1 Требования к программе 2](#_Toc69058705)

[1.1 Задание 2](#_Toc69058706)

[1.2 Исходные данные 2](#_Toc69058707)

[1.3 Функциональные требования к программе 3](#_Toc69058708)

[1.4 Требования к программной реализации 5](#_Toc69058709)

[2 Конструирование программы 7](#_Toc69058710)

[2.1 Разработка модульной структуры программы 7](#_Toc69058711)

[2.2 Выбор способа организации данных 7](#_Toc69058712)

[2.3 Разработка перечня пользовательских функций программы 8](#_Toc69058712)

[3 Разработка алгоритмов программы](#_Toc69058713) 11

[3.1 Алгоритм функции main 11](#_Toc69058714)

[3.2 Алгоритм функции bool deleteYourselfAccount(vector <Accounts>& vector\_of\_accs, string your\_login); 11](#_Toc69058715)

[3.3 Алгоритм функции char choiceAction(char begin, char end); 12](#_Toc69058716)

[4 Описание работы программы 12](#_Toc69058713)

[4.1 Авторизация 12](#_Toc69058714)

[4.2 Модуль администратора 13](#_Toc69058715)

[4.3 Модуль пользователя 14](#_Toc69058716)

[4.4 Исключительные ситуации 15](#_Toc69058714)

[ПРИЛОЖЕНИЕ 16](#_Toc69058717)

1 ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1 Задание

Разработка программы планирования бюджета проекта

В компании по разработке программного обеспечения для планирования работ по проекту необходимо обработать информацию следующего вида: наименование проекта, вид работ (работа над требованиями, разработка архитектуры, реализация, тестирование), ФИО сотрудника, предполагаемое количество часов, стоимость одного часа.

Индивидуальное задание: для каждого проекта вывести его итоговую стоимость, перечень видов работ с указанием общего количества задействованных специалистов и стоимости данного этапа.

Общее для всех вариантов задание: реализовать авторизацию для входа в систему, функционал администратора и функционал пользователя (см. более подробно в функциональных требованиях к курсовой работе).

1.2 Исходные данные

Тема курсовой работы выбирается из списка, приведенного в

приложении 1.

Язык программирования С++.

Среда разработки Microsoft Visual Studio.

Вид приложения – консольное.

Парадигма программирования – процедурная (по согласованию с преподавателем допускается реализация программы в рамках объектно-ориентированной парадигмы программирования).

Способ организации данных – структуры (struct) (либо поля классов в случае объектно-ориентированного программирования).

Способ хранения данных – файлы (по согласованию с преподавателем допускается подключение баз данных).

Каждая логически завершенная подзадача программы должна быть реализована в виде отдельной функции (метода в случае объектно-ориентированного программирования).

Построение программного кода должно соответствовать соглашению о коде «С++ Code Convention».

К защите курсовой работы представляются: консольное приложение и пояснительная записка.

Текст пояснительной записки оформляется в соответствии со стандартом предприятия СТП 01–2017.

1.3 Функциональные требования к программе

Данные аккаунтов и сотрудников хранятся в отдельных файлах в форме соответствующих структур.

При каждом вводе пароля, он маскируется звездочками.

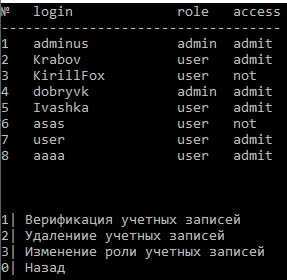
Алгоритм регистрации устроен через подтверждение заявки на создание нового аккаунта в системе администратором. В момент оформления новой заявки запрашиваемый логин проверяется на уникальность и, в случае наличия аккаунта с таким логином в системе, выдает пользователю сообщение и выходит в главное меню.

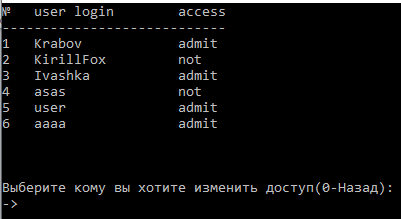
Все пароли хешируются через sha256 с добавлением “соли” и в файле хранится только хеш соответствующего пароля и его “соль”.

Работа с данными:

Модуль администратора включает два подмодуля (работа с учетными записями и данными самой системы).

В первом реализованы следующие функции (рис. 1.1):





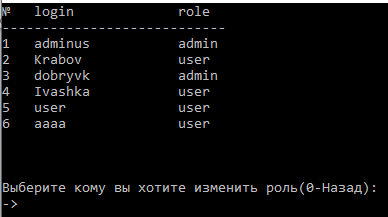


Рисунок 1.1

Во втором реализованы следующие функции (рис. 1.2):

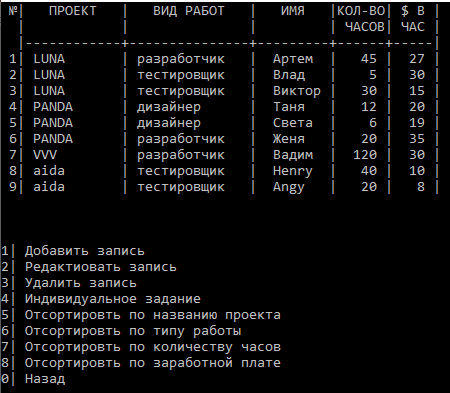


Рисунок 1.2

В модуле пользователя также реализованы данные(рисунок 1.2) функции, кроме добавления/удаления/редактирования данных студентов

В работе также предусмотрена обработка исключительных ситуаций, навигация, запрос на выполнение необратимых действий, обратная связь с пользователем.

1.4 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\_accounts) или «верблюжья нотация» (например, flagExit). Константам рекомендуется присваивать имена, состоящие из букв верхнего регистра (например, FILE\_OF\_ACCOUNTS).

2. Имена функций должны быть осмысленными, начинаться с буквы нижнего регистра, строится по принципу глагол+существительное (например,

addAccount, sortWorkersByCompany). 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).

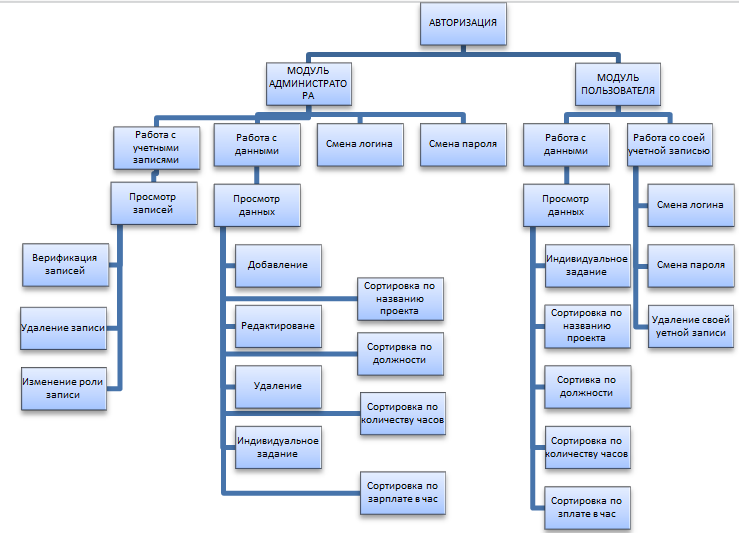
9. Следует избегать глубокой вложенности условных и циклических конструкций: вложенность блоков должна быть не более трѐх.

10. Следует избегать длинных функций: текст функции должен умещаться на один экран (в эквиваленте строк – не превышать 25-50 строчек).

11. Следует выносить 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



2.2 Выбор способа организации данных

// Структура для аккаунта.

struct Accounts

{

string login;//логин аккаунта

string salt;//соль для хеширование

string salted\_hash\_password;//захешированный пароль с солью

int role;//роль,1-админ 0-юзер

int access;//доступ,1-разрешен 0-нет

};

// Стурктура для сотрудника.

struct Workers

{

string company;//проект

string work;//должность

string name;//имя

int hours;//рабочие часы

int money;//зарплата в час

};

Способ объединения входных данных – вектор.

2.3 Разработка перечня пользовательских функций программы

validation.h.

//проверка строки на валидность

bool checkString(string str);

//запрос на ввод целого числа из диапазона

int getInt(int bein, int end);

//выбор действия

char choiceAction(char begin, char end);

//подтверждения необратимого действия

bool confirmAction(string str);

//ввод имени и должности

string inputName();

//ввод названия проека

string inputCompany();

interfaces.h.

//получение количества структур

int getCountOfStructuresInFile(string file\_path);

//вывод информации в конце

void inputInfoAboutProgram();

//вывод основного меню

char outputMainMenu();

//основной интерфейс

void System(vector <Accounts>& vector\_of\_accs, vector <Workers>& vec\_of\_workers);

//админ панель

void adminInterface(vector <Accounts>& vector\_of\_accs, vector <Workers>& vector\_of\_workers,string login);

//вывести админ меню

char outputAdminMenu(string login);

//работа с аккаунтами для админа

void workingWithAccounts(vector <Accounts>& vector\_of\_accs,string login);

//вывод меню

char outputWorkingWithAccountsMenu();

//работа с даными для админа

void workingWithWorkersAdmin(vector <Workers>& vector\_of\_workers);

//вывод меню

char outputWorkingWithWorkersMenu();

//юзер панель

void userInterface(vector <Accounts>& vector\_of\_accs, vector <Workers>& vector\_of\_workers, string login);

//вывод меню

char outputUserMenu(string login);

//работа с данными для юзера

bool workingWithAccountByUser(vector <Accounts>& vector\_of\_accs, string login);

//вывод мен.

char outputWorkingWithUserAccountMenu();

//работа со своей уч аписью для юзера

void workingWithWorkersUser(vector <Workers>& vector\_of\_workers);

//вывод меню

char outputWorkingWithWorkersMenuUser();

accounts.h.

//считать инфу из файла

void readFileAccounts(vector <Accounts>& vector\_of\_accs);

//записать инфу в файл

void writeFileAccounts(const vector <Accounts>& vector\_of\_accs);

//ввод пароль

string inputPass();

//регистрация аккаунта

void registeringAccount(vector <Accounts>& vector\_of\_accs);

//вход в аккаунт

string loginAccount(const vector <Accounts>& vector\_of\_accs, int& role);

//смена логина

string changeLogin( vector <Accounts>& vector\_of\_accs,string login);

//смена пароля

void changePassword(vector <Accounts>& vector\_of\_accs, string login);

//вывод всех аккаунтов

void outputAllAccounts(vector <Accounts>& vector\_of\_accs);

//верификация аккаунта(смена роль)

void verificationAccount(vector <Accounts>& vector\_of\_accs);

//удаления аккаунта юзера

void deleteUserAccount(vector <Accounts>& vector\_of\_accs,string your\_login);

//смена роли аккаунту

void changeRoleAccount(vector <Accounts>& vector\_of\_accs,string your\_login);

//удаление себя юзером

bool deleteYourselfAccount(vector <Accounts>& vector\_of\_accs, string your\_login);

workers.h.

//считать из файла

void readFileWorkers(vector <Workers>& vector\_of\_workers);

//записать в файл

void writeFileWorkers(const vector <Workers>& vector\_of\_workers);

//вывод всех данных

void outputAllWorkers(const vector <Workers>& vector\_of\_workers);

//добавить сотрудника

void addWorker(vector <Workers>& vector\_of\_workers);

//редактировать

void editWorker(vector <Workers>& vector\_of\_workers);

//удалить

void deleteWorker(vector <Workers>& vector\_of\_workers);

//сортировки

void sortByCompany(vector <Workers>& vector\_of\_workers);

void sortByWork(vector <Workers>& vector\_of\_workers);

void sortByHours(vector <Workers>& vector\_of\_workers);

void sortByMoney(vector <Workers>& vector\_of\_workers);

//компораторы к сортировкам

bool mySortByCompany(Workers a, Workers b);

bool mySortByWork(Workers a, Workers b);

bool mySortByHours(Workers a, Workers b);

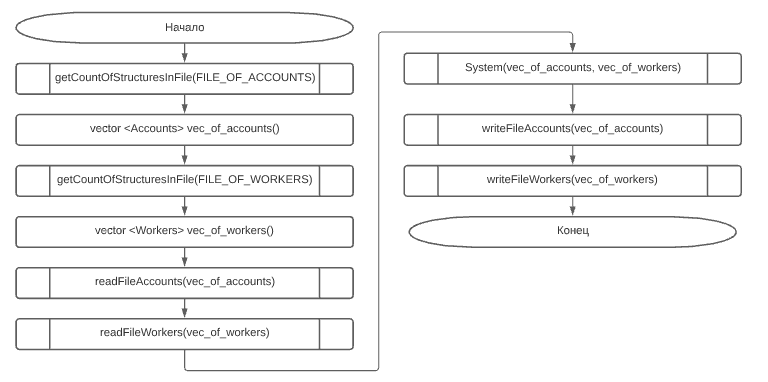
bool mySortByMoney(Workers a, Workers b);

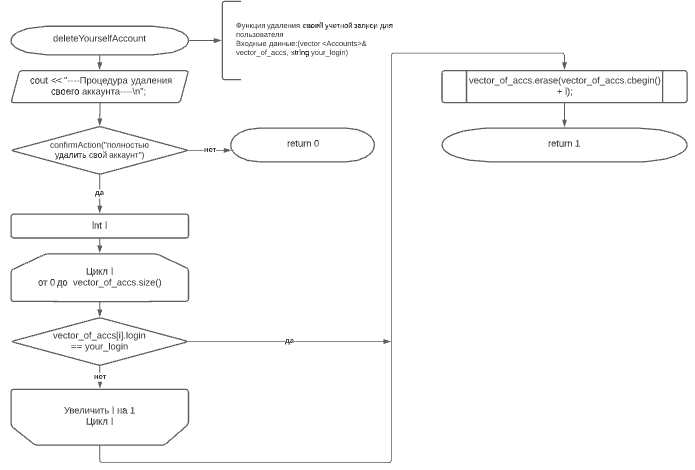
//индивидуальное задание

void individualTask(vector <Workers>& main\_vector);

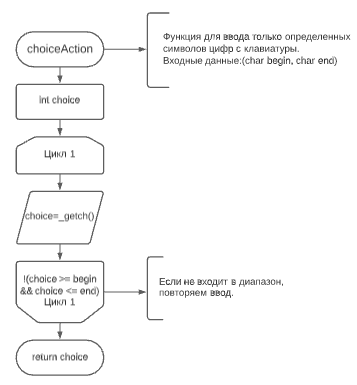
3 РАЗРАБОТКА АЛГОРИТМОВ ПРОГРАММЫ

3.1 Алгоритм функции main



3.2 Алгоритм функции bool deleteYourselfAccount(vector <Accounts>& vector\_of\_accs, string your\_login)

3.3 Алгоритм char choiceAction(char begin, char end)



4 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

4.1 Авторизация

В начале авторизации программа предлагает пользователю войти, зарегистрировать новый аккаунт или выйти (рис. 4.1).

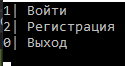


Рисунок 4.1

В случае регистрации будет создана новая учетная запись без доступа к системе. Доступ будет открыт как только администратор одобрит заявку на регистрацию нового аккаунта. Если же пользователь уже имеет учетную запись с доступом к системе он может авторизоваться используя свой логин и пароль. Хеш введенного пароля сверяется с тем, что хранится в файле, и, в случае их совпадения, авторизация завершается успешно.

4.2 Модуль администратора

Модуль администратора включает в себя работу с учетными записями, работу с данными и смену логина и пароля (рис. 4.2).

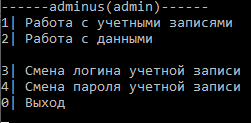


Рисунок 4.2

Сами подмодули реализованы в удобном для просмотра ключе (рис. 4.3 и рис. 4.4).

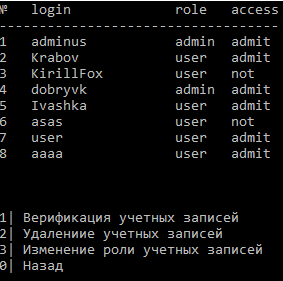


Рисунок 4.4

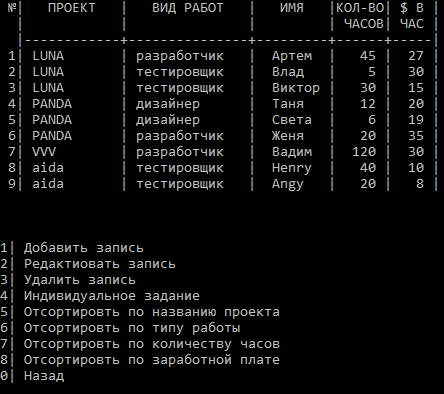


Рисунок 4.5

4.3 Модуль пользователя

Модуль пользователя включает в себя работу с данными и работу со своей учетной записью(рис. 4.6).

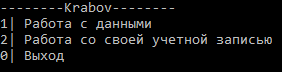


Рисунок 4.6

В подмодуле работы с данными пользователь может просмотреть данные, просмотреть на выполнение индивидуального задания ,а также выполнить различные сортировки(рис. 4.7)

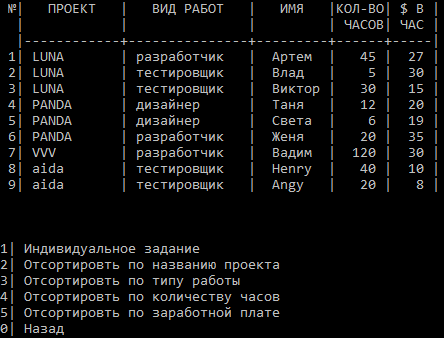


Рисунок 4.7

В подмодуле работы со своей учетной записью пользователь может изменить логин или пароль своего аккаунт, а также удалить свой аккаунт навсегда(после чего его направит в главное меню программы)(рис. 4.8)

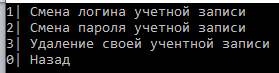


Рисунок 4.8

4.4 Исключительные ситуации

На этапе авторизации предусмотрены ситуации некорректного ввода данных аккаунта, попытка регистрации нового аккаунта с логином, который уже используется.

Корректный ввод числовых данных реализован в каждой функции программы, учитывая все логичные значения.

Запрос на выполнение необратимых действий исключает кейсы случайных ошибочных действий.

Предусмотрен случай отсутствия текстовых файлов.

Проверка строки на валидные символы.

ПРИЛОЖЕНИЕ

**Source.cpp**

#include <vector>

#include <windows.h>

#include "accounts.h"

#include "workers.h"

#include "interfaces.h"

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

vector <Accounts> vec\_of\_accounts(getCountOfStructuresInFile(FILE\_OF\_ACCOUNTS));

vector <Workers> vec\_of\_workers(getCountOfStructuresInFile(FILE\_OF\_WORKERS));

readFileAccounts(vec\_of\_accounts);

readFileWorkers(vec\_of\_workers);

System(vec\_of\_accounts, vec\_of\_workers);

writeFileAccounts(vec\_of\_accounts);

writeFileWorkers(vec\_of\_workers);

return 0;

}

**accounts.h**

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include <conio.h>

#include <fstream>

#include <iomanip>

#include <cstdlib>

#include <ctime>

#include "sha256.h"

#include "validation.h"

using namespace std;

struct Accounts

{

string login;//логин аккаунта

string salt;//соль для хеширование

string salted\_hash\_password;//захешированный пароль с солью

int role;//роль,1-админ 0-юзер

int access;//доступ,1-разрешен 0-нет

};

const string FILE\_OF\_ACCOUNTS = "AccData.txt";

const int SALT\_SIZE= 12;

const int SYMBOLS\_SIZE = 62;

void readFileAccounts(vector <Accounts>& vector\_of\_accs);

void writeFileAccounts(const vector <Accounts>& vector\_of\_accs);

string inputPass();

void registeringAccount(vector <Accounts>& vector\_of\_accs);

string loginAccount(const vector <Accounts>& vector\_of\_accs, int& role);

string changeLogin( vector <Accounts>& vector\_of\_accs,string login);

void changePassword(vector <Accounts>& vector\_of\_accs, string login);

void outputAllAccounts(vector <Accounts>& vector\_of\_accs);

void verificationAccount(vector <Accounts>& vector\_of\_accs);

void deleteUserAccount(vector <Accounts>& vector\_of\_accs,string your\_login);

void changeRoleAccount(vector <Accounts>& vector\_of\_accs,string your\_login);

bool deleteYourselfAccount(vector <Accounts>& vector\_of\_accs, string your\_login);

string generateSalt(int salt\_size);

string getSymbolsForSalt();

**accounts.cpp**

#include "accounts.h"

void readFileAccounts(vector <Accounts>& vector\_of\_accs)

//считаем из файла данные об аккаунтах(если его нет - создадим)

{

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

if ((!fin.is\_open()) || vector\_of\_accs.empty())

{

ofstream fout(FILE\_OF\_ACCOUNTS);

fout.close();

string salt = generateSalt(SALT\_SIZE);

string salted\_hash\_password = sha256("adminus" + salt);

Accounts temp;

temp.login = "adminus";

temp.salt = salt;

temp.salted\_hash\_password = salted\_hash\_password;

temp.role = 1;

temp.access = 1;

vector\_of\_accs.push\_back(temp);

}

else

{

int i = 0;

while (!fin.eof())

{

fin >> vector\_of\_accs[i].login

>> vector\_of\_accs[i].salt

>> vector\_of\_accs[i].salted\_hash\_password

>> vector\_of\_accs[i].role

>> vector\_of\_accs[i].access;

i++;

}

}

fin.close(); //Закрыли файл

}

void writeFileAccounts(const vector <Accounts>& vector\_of\_accs)

//запишем в файл данные об аккаунтах

{

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

for (int i = 0; i < vector\_of\_accs.size(); i++)

{

fout << vector\_of\_accs[i].login << " "

<< vector\_of\_accs[i].salt << " "

<< vector\_of\_accs[i].salted\_hash\_password << " "

<< vector\_of\_accs[i].role << " "

<< vector\_of\_accs[i].access;

if (i < vector\_of\_accs.size() - 1)

fout << endl;

}

fout.close();

}

string inputPass() // ф-я ввода пароля

//ввод пароля со \*

{

string pass;

int symbol;

while (true)

{

symbol = \_getch();// забираем символ с консоли и не выводим

if (symbol == 13) // если код клавиши enter то переводим строку и выходим из цикла

{

cout << endl;

break;

}

else

{

if (symbol == 8) // если код клавиши backspace то:1 если строка пуста ничего не делаем

{

if (!pass.empty()) //2 если чтото есть:убираем последний символ в строке и передвигаем курсор влево, заменяя \* пробелом и снова смещаем

{

pass.erase(pass.length() - 1);

cout << (char)8 << ' ' << (char)8;

}

}

else

{

cout << '\*';

pass += (char)symbol;

}

}

}

return pass;

}

void registeringAccount(vector <Accounts>& vector\_of\_accs)

//регистрация акка с защитами

{

string login, password, password2;

bool flagLogin = 1, flagTotalPass = 1;

cout << "Регистрация аккаунта:\n";

while (flagLogin)

{

cout << "Придумайте логин:\n-> ";

cin >> login;

flagLogin = checkString(login);

for (int i = 0; i < vector\_of\_accs.size(); i++)

{

if (vector\_of\_accs[i].login == login)

{

cout << "Аккаунт с таким логином уже существует,повторите попытку с другим логином.\n";

flagLogin = 1;

}

}

}

while (flagTotalPass)

{

bool flagPass = 1;

while (flagPass)

{

cout << "Придумайте пароль:\n-> ";

password = inputPass();

flagPass = checkString(password);

}

flagPass = 1;

while (flagPass)

{

cout << "Повторите пароль:\n-> ";

password2 = inputPass();

flagPass = checkString(password);

}

if (password == password2)

{

flagTotalPass = 0;

}

else

{

cout << "Ошибка:пароли не совпадают.Попробуйте еще.\n";

}

}

cout << "Аккаунт зарегистрирован! Ожидайте подтверждения администратором.\n";

string salt = generateSalt(SALT\_SIZE);

string salted\_hash\_password = sha256(password + salt);

Accounts temp;

temp.login = login;

temp.salt = salt;

temp.salted\_hash\_password = salted\_hash\_password;

temp.role = 0;

temp.access = 0;

vector\_of\_accs.push\_back(temp);

\_getch();

system("cls");

}

string loginAccount(const vector <Accounts>& vector\_of\_accs, int& role)

//ф-я вхождения в аккаунт 1-вход как админ 0-вход как юзер (-1)-вход не выполнен

{

string login, password;

cout << "Введите логин:\n-> ";

cin >> login;

cout << "Введите пароль\n-> ";

password = inputPass();

for (int i = 0; i < vector\_of\_accs.size(); i++)

if (vector\_of\_accs[i].login == login)

{

if (vector\_of\_accs[i].salted\_hash\_password == sha256(password + vector\_of\_accs[i].salt))

{

if (vector\_of\_accs[i].access == 0)

{

cout << "Аккаунт ожидает верификации или он забанен.\n";

\_getch();

return "";

}

else

{

cout << "Вы вошли как " + login;

if (vector\_of\_accs[i].role == 1)

{

cout << "(admin)\n";

role = 1;

}

else

{

cout << endl;

role = 0;

}

return login;

;

}

}

else

break;

}

cout << "Не удается войти.\nПожалуйста, проверьте правильность введенных данных.\n";

return "";

}

string changeLogin(vector <Accounts>& vector\_of\_accs, string login)

//смена логина

{

cout << "----Система смены логина----\n";

cout << "Ваш текущий логин:\n" << login << endl;

string new\_login;

do

{

cout << "Введите ваш новый логин:\n-> ";

cin >> new\_login;

} while (checkString(new\_login));

for (int i = 0; i < vector\_of\_accs.size(); i++)

if (vector\_of\_accs[i].login == login)

{

vector\_of\_accs[i].login = new\_login;

break;

}

cout << "Логин успешно сменен на " + new\_login << endl;

\_getch();

return new\_login;

}

void changePassword(vector <Accounts>& vector\_of\_accs, string login)

//смена пароля

{

cout << "----Система смены пароля----\n";

cout << "Введите ваш текущий пароль:\n-> ";

string password = inputPass();

int cell\_number;

for (cell\_number = 0; cell\_number < vector\_of\_accs.size(); cell\_number++) //находим номер ячейки где храняться данные этого акка

{

if (vector\_of\_accs[cell\_number].login == login)

break;

}

if (sha256(password + vector\_of\_accs[cell\_number].salt) == vector\_of\_accs[cell\_number].salted\_hash\_password)

{

bool flagTotalPass = 1;

string password2;

while (flagTotalPass)

{

bool flagPass = 1;

while (flagPass)

{

cout << "Придумайте новый пароль:\n-> ";

password = inputPass();

flagPass = checkString(password);

}

flagPass = 1;

while (flagPass)

{

cout << "Повторите новый пароль:\n-> ";

password2 = inputPass();

flagPass = checkString(password);

}

if (password == password2)

flagTotalPass = 0;

else

cout << "Ошибка:пароли не совпадают.Попробуйте еще.\n";

}

string new\_salt = generateSalt(SALT\_SIZE);

string new\_salted\_hash\_password = sha256(password + new\_salt);

vector\_of\_accs[cell\_number].salt = new\_salt;

vector\_of\_accs[cell\_number].salted\_hash\_password = new\_salted\_hash\_password;

cout << "Пароль успешно сменен.\n";

}

else

{

cout << "Пароль неверен!\n";

}

\_getch();

}

void verificationAccount(vector <Accounts>& vector\_of\_accs)

//ф-я разрешения/запрета доступа

{

while (true)

{

cout << left << setw(4) << "№" << left << setw(18) << "user login" << left << setw(6) << "access" << endl;

cout << "----------------------------\n";

vector <string> temp\_vector;

int number\_user\_accs = 0;

for (int i = 0; i < vector\_of\_accs.size(); i++)

{

if (vector\_of\_accs[i].role == 0)

{

temp\_vector.push\_back(vector\_of\_accs[i].login);

number\_user\_accs++;

cout << left << setw(4) << number\_user\_accs << left << setw(18) << vector\_of\_accs[i].login;

if (vector\_of\_accs[i].access == 1)

cout << left << setw(6) << "admit";

else

cout << left << setw(6) << "not";

cout << endl;

}

}

if (number\_user\_accs == 0)

{

cout << "Список пользователей пуст!\n";

\_getch();

system("cls");

return;

}

cout << "\n\n\n";

cout << "Выберите кому вы хотите изменить доступ(0-Назад):\n-> ";

int choice = getInt(0, number\_user\_accs);

if (choice == 0)

return;

string changing\_access\_login = temp\_vector[choice - 1];

for (int i = 0; i < vector\_of\_accs.size(); i++)

{

if (vector\_of\_accs[i].login == changing\_access\_login)

{

if (vector\_of\_accs[i].access == 1)

vector\_of\_accs[i].access = 0;

else

vector\_of\_accs[i].access = 1;

break;

}

}

system("cls");

}

}

void outputAllAccounts(vector <Accounts>& vector\_of\_accs)

//вывод всех акков

{

cout << left << setw(4) << "№" << left << setw(18) << "login" << left << setw(7) << "role" << left << setw(6) << "access" << endl;

cout << "-----------------------------------\n";

for (int i = 0; i < vector\_of\_accs.size(); i++)

{

cout << left << setw(4) << i + 1 << left << setw(18) << vector\_of\_accs[i].login;

if (vector\_of\_accs[i].role == 1)

cout << left << setw(7) << "admin";

else

cout << left << setw(7) << "user";

if (vector\_of\_accs[i].access == 1)

cout << left << setw(6) << "admit";

else

cout << left << setw(6) << "not";

cout << endl;

}

cout << "\n\n\n";

}

void deleteUserAccount(vector <Accounts>& vector\_of\_accs, string your\_login)

//удаление акка юзера

{

outputAllAccounts(vector\_of\_accs);

cout << "Выберите, номер удаляемого пользователя(0-Назад):\n-> ";

int choice = getInt(0, vector\_of\_accs.size());

if (choice == 0)

return;

choice--;

if (vector\_of\_accs[choice].login == your\_login)

{

cout << "Нельзя удалять самого себя(((\n";

\_getch();

return;

}

if (vector\_of\_accs[choice].role == 1)

{

cout << "Нельзя удалять админов.\n";

\_getch();

return;

}

if (confirmAction("удалить пользователя " + vector\_of\_accs[choice].login))

{

cout << "Пользователь " << vector\_of\_accs[choice].login << " успешно удален.\n";

\_getch();

vector\_of\_accs.erase(vector\_of\_accs.cbegin() + choice);

}

}

void changeRoleAccount(vector <Accounts>& vector\_of\_accs, string your\_login)

//изменение пароля

{

while (true)

{

cout << left << setw(4) << "№" << left << setw(18) << "login" << left << setw(6) << "role" << endl;

cout << "----------------------------\n";

vector <string> temp\_vector;

int number\_admit\_accs = 0;

for (int i = 0; i < vector\_of\_accs.size(); i++)

{

if (vector\_of\_accs[i].access == 1)

{

temp\_vector.push\_back(vector\_of\_accs[i].login);

number\_admit\_accs++;

cout << left << setw(4) << number\_admit\_accs << left << setw(18) << vector\_of\_accs[i].login;

if (vector\_of\_accs[i].role == 1)

cout << left << setw(6) << "admin";

else

cout << left << setw(6) << "user";

cout << endl;

}

}

cout << "\n\n\n";

cout << "Выберите кому вы хотите изменить роль(0-Назад):\n-> ";

int choice = getInt(0, number\_admit\_accs);

if (choice == 0)

return;

if (temp\_vector[choice - 1] == your\_login)

{

cout << "Вы не можете изменить себе роль!\n";

\_getch();

continue;

}

string changing\_role\_login = temp\_vector[choice - 1];

for (int i = 0; i < vector\_of\_accs.size(); i++)

{

if (vector\_of\_accs[i].login == changing\_role\_login)

{

if (vector\_of\_accs[i].role == 1)

vector\_of\_accs[i].role = 0;

else

vector\_of\_accs[i].role = 1;

break;

}

}

system("cls");

}

}

bool deleteYourselfAccount(vector <Accounts>& vector\_of\_accs, string your\_login)

//ф-я удаления своего акка для юзера

{

//вернет 0-если все хорошо 1-если аккаунт удален и требуется выйти к основному меню

cout << "----Процедура удаления своего аккаунта----\n";

if (confirmAction("полностью удалить свой аккаунт"))

{

int i;

for (i = 0; i < vector\_of\_accs.size(); i++)

if (vector\_of\_accs[i].login == your\_login)

break;

vector\_of\_accs.erase(vector\_of\_accs.cbegin() + i);

return 1;

}

return 0;

}

string generateSalt(int salt\_size)

//генерация соли

{

string symbols = getSymbolsForSalt();

srand(time(0));

string salt;

salt.reserve(salt\_size);

for (int i = 0; i < salt\_size; i++)

salt.push\_back(symbols[rand() % SYMBOLS\_SIZE]);

return salt;

}

string getSymbolsForSalt()

{

string symbols;

symbols.reserve(SYMBOLS\_SIZE);

char little\_latter = 'a';

char big\_latter = 'A';

char number = '0';

int i;

for (int j = 0; j < 26; j++)

{

symbols.push\_back(little\_latter++);

symbols.push\_back(big\_latter++);

if (j < 10)symbols.push\_back(number++);

}

return symbols;

}

**workers.h**

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include <fstream>

#include <iomanip>

#include <conio.h>

#include <algorithm>

#include "validation.h"

using namespace std;

struct Workers

{

string company;//проект

string work;//должность

string name;//имя

int hours;//рабочие часы

int money;//зарплата в час

};

const string FILE\_OF\_WORKERS = "Data.txt";

void readFileWorkers(vector <Workers>& vector\_of\_workers);

void writeFileWorkers(const vector <Workers>& vector\_of\_workers);

void outputAllWorkers(const vector <Workers>& vector\_of\_workers);

void addWorker(vector <Workers>& vector\_of\_workers);

void editWorker(vector <Workers>& vector\_of\_workers);

void deleteWorker(vector <Workers>& vector\_of\_workers);

void sortByCompany(vector <Workers>& vector\_of\_workers);

void sortByWork(vector <Workers>& vector\_of\_workers);

void sortByHours(vector <Workers>& vector\_of\_workers);

void sortByMoney(vector <Workers>& vector\_of\_workers);

bool mySortByCompany(Workers a, Workers b);

bool mySortByWork(Workers a, Workers b);

bool mySortByHours(Workers a, Workers b);

bool mySortByMoney(Workers a, Workers b);

void individualTask(vector <Workers>& main\_vector);

**workers.cpp**

#include "workers.h"

void readFileWorkers(vector <Workers>& vector\_of\_workers)

//ф-я считывания данных из файла(при необходимости его создание)

{

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

if ((!fin.is\_open()) || vector\_of\_workers.empty())

{

ofstream fout(FILE\_OF\_WORKERS);

fout.close();

}

else

{

int i = 0;

while (!fin.eof())

{

fin >> vector\_of\_workers[i].company

>> vector\_of\_workers[i].work

>> vector\_of\_workers[i].name

>> vector\_of\_workers[i].hours

>> vector\_of\_workers[i].money;

i++;

}

}

fin.close(); //Закрыли файл

}

void writeFileWorkers(const vector <Workers>& vector\_of\_workers)

//ф-я для записи данных в файл

{

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

for (unsigned int i = 0; i < vector\_of\_workers.size(); i++)

{

fout << vector\_of\_workers[i].company << " "

<< vector\_of\_workers[i].work << " "

<< vector\_of\_workers[i].name << " "

<< vector\_of\_workers[i].hours << " "

<< vector\_of\_workers[i].money;

if (i < vector\_of\_workers.size() - 1)

fout << endl;

}

fout.close();

}

void outputAllWorkers(const vector <Workers>& vector\_of\_workers)

//ф-я вывода таблицы с информацией о работниках

{

cout << " №| ПРОЕКТ | ВИД РАБОТ | ИМЯ |КОЛ-ВО| $ В |\n";

cout << " | | | | ЧАСОВ| ЧАС |\n";

cout << " |------------+---------------+---------+------+-----|\n";

int size = vector\_of\_workers.size();

if (size == 0)

{

cout << "Список записей пуст!\n";

cout << "\n\n\n";

return;

}

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

{

cout << setw(2) << i + 1 <<

"| " << left << setw(11) << vector\_of\_workers[i].company

<< "| " << left << setw(14) << vector\_of\_workers[i].work

<< "| " << left << setw(7) << vector\_of\_workers[i].name

<< "| " << right << setw(3) << vector\_of\_workers[i].hours

<< " | " << right << setw(3) << vector\_of\_workers[i].money

<< " |\n";

}

cout << "\n\n\n";

}

void addWorker(vector <Workers>& vector\_of\_workers)

//ф-я добавления работника в вектор

{

string company, work, name;

int hours, money;

cout << "----Добавление записи----\n";

cout << "Введите название проекта:\n-> ";

company = inputCompany();

cout << "Введите должность работника:\n-> ";

work = inputName();

cout << "Введите имя работника:\n-> ";

name = inputName();

cout << "Введите количество часов работника:\n-> ";

hours = getInt(0, 350);

cout << "Введите почасовую оплату в $:\n-> ";

money = getInt(0, 500);

cout << "Вы успешно добавили запись!\n";

Workers temp;

temp.company = company;

temp.work = work;

temp.name = name;

temp.hours = hours;

temp.money = money;

vector\_of\_workers.push\_back(temp);

\_getch();

}

void editWorker(vector <Workers>& vector\_of\_workers)

//ф-я редактирования работника

{

outputAllWorkers(vector\_of\_workers);

cout << "Выберите, номер редактируемого работника(0-Назад):\n-> ";

int choice = getInt(0, vector\_of\_workers.size());

if (choice == 0)

return;

choice--;

cout << "--Что редактируем?--\n";

cout << "1| Проект\n";

cout << "2| Должность\n";

cout << "3| Имя\n";

cout << "4| Количество рабочих часов\n";

cout << "5| Оплата за рабочий час\n";

cout << "0| Отмена\n";

switch (choiceAction('0', '5'))

{

case '0':

return;

case '1':

{

cout << "Введите новое название проекта:\n-> ";

string company = inputCompany();

vector\_of\_workers[choice].company = company;

break;

}

case '2':

{

cout << "Введите новую должность работника:\n-> ";

string work = inputName();

vector\_of\_workers[choice].work = work;

break;

}

case '3':

{

cout << "Введите имя нового работника:\n-> ";

string name = inputName();

vector\_of\_workers[choice].name = name;

break;

}

case '4':

{

cout << "Введите новое количество рабочих часов работника:\n-> ";

int hours = getInt(0, 350);

vector\_of\_workers[choice].hours = hours;

break;

}

case '5':

{

cout << "Введите новую заработную плату в час в $:\n-> ";

int money = getInt(0, 500);

vector\_of\_workers[choice].money = money;

break;

}

}

cout << "Успешно изменено.\n";

\_getch();

}

void deleteWorker(vector <Workers>& vector\_of\_workers)

{

//функция удаления работника из вектора

outputAllWorkers(vector\_of\_workers);

cout << "Выберите, номер удаляемого работника(0-Назад):\n-> ";

int choice = getInt(0, vector\_of\_workers.size());

if (choice == 0)

return;

choice--;

if (confirmAction("удалить пользователя " + vector\_of\_workers[choice].name))

{

cout << "Запись " << vector\_of\_workers[choice].name << " успешно удален.\n";

\_getch();

vector\_of\_workers.erase(vector\_of\_workers.cbegin() + choice);

}

}

void sortByCompany(vector <Workers>& vector\_of\_workers)

//сортировка по названию компании

{

sort(vector\_of\_workers.begin(), vector\_of\_workers.end(), mySortByCompany);

}

void sortByWork(vector <Workers>& vector\_of\_workers)

//сортировка по названию должности

{

sort(vector\_of\_workers.begin(), vector\_of\_workers.end(), mySortByWork);

}

void sortByHours(vector <Workers>& vector\_of\_workers)

//сортировка по рабочим часам

{

sort(vector\_of\_workers.begin(), vector\_of\_workers.end(), mySortByHours);

}

void sortByMoney(vector <Workers>& vector\_of\_workers)

//сортировка по зарплате

{

sort(vector\_of\_workers.begin(), vector\_of\_workers.end(), mySortByMoney);

}

bool mySortByCompany(Workers a, Workers b)

{

return a.company < b.company;

}

bool mySortByWork(Workers a, Workers b)

{

return a.work < b.work;

}

bool mySortByHours(Workers a, Workers b)

{

return a.hours > b.hours;

}

bool mySortByMoney(Workers a, Workers b)

{

return a.money > b.money;

}

void individualTask(vector <Workers>& main\_vector)

//ф-я выполнения индивидуального задания

{

if (main\_vector.empty())

return;

sortByWork(main\_vector);

sortByCompany(main\_vector);

int vec\_size = main\_vector.size();

int iterator = 0;

while (iterator < vec\_size)

{

string i\_company = main\_vector[iterator].company;

int total\_cost = 0, i = iterator;

while (i\_company == main\_vector[i].company) //подсчет общего бюджета

{

total\_cost += main\_vector[i].hours \* main\_vector[i].money;

i++;

if (i == vec\_size)//если последний элемент

break;

}

cout << endl << i\_company << endl << endl;

cout << "Общая стоимость проекта: " << total\_cost << "$\n";

cout << "| ВИД РАБОТ | КОЛИЧЕСТВО | СТОИМОСТЬ |\n";

cout << "| |СПЕЦИАЛИСТОВ| |\n";

while (iterator < i)

{

string i\_work = main\_vector[iterator].work;

int stage\_cost = 0, num\_of\_employees = 0;

while (i\_work == main\_vector[iterator].work)// подсчет бюджета на отрасль

{

int hours = main\_vector[iterator].hours, money = main\_vector[iterator].money;

stage\_cost +=hours\* money;

num\_of\_employees++;

iterator++;

if (iterator == vec\_size)

break;

}

cout << "| " << left << setw(14) << i\_work << "| " << right << setw(2) << num\_of\_employees << " | " << right << setw(5)<<stage\_cost << " |\n";

}

cout << "\n------------------------------------------------------------\n";

}

cout << "Нажмите, чтобы вернуться обратно.\n";

\_getch();

}

**Interfaces.h**

#pragma once

#include <string>

#include <vector>

#include <iomanip>

#include "validation.h"

#include "accounts.h"

#include "workers.h"

int getCountOfStructuresInFile(string file\_path);

void inputInfoAboutProgram();

char outputMainMenu();

void System(vector <Accounts>& vector\_of\_accs, vector <Workers>& vec\_of\_workers);

void adminInterface(vector <Accounts>& vector\_of\_accs, vector <Workers>& vector\_of\_workers,string login);

char outputAdminMenu(string login);

void workingWithAccounts(vector <Accounts>& vector\_of\_accs,string login);

char outputWorkingWithAccountsMenu();

void workingWithWorkersAdmin(vector <Workers>& vector\_of\_workers);

char outputWorkingWithWorkersMenu();

void userInterface(vector <Accounts>& vector\_of\_accs, vector <Workers>& vector\_of\_workers, string login);

char outputUserMenu(string login);

bool workingWithAccountByUser(vector <Accounts>& vector\_of\_accs, string login);

char outputWorkingWithUserAccountMenu();

void workingWithWorkersUser(vector <Workers>& vector\_of\_workers);

char outputWorkingWithWorkersMenuUser();

**interfaces.cpp**

#include "interfaces.h"

int getCountOfStructuresInFile(string file\_path)

// Определение количества структур в файле

{

ifstream file(file\_path, ios::in); // Открыли текстовый файл для чтения

int number\_of\_strings = 0;

if (file.is\_open())

{

if (file.peek() == EOF)

return 0;

while (file.ignore(numeric\_limits<streamsize>::max(), '\n'))

number\_of\_strings++;

}

file.close();

return number\_of\_strings;

}

char outputMainMenu()

//вывод основной менюшки и выбор

{

cout << "1| Войти\n";

cout << "2| Регистрация\n";

cout << "0| Выход\n";

return choiceAction('0', '2');

}

void inputInfoAboutProgram()

//конечная ,просто инфа

{

cout << "\n\n\n\n";

cout << " БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ\n";

cout << " ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ\n\n\n\n";

cout << " ОСНОВЫ КОНСТРУИРОВАНИЯ ПРОГРАММ\n\n\n";

cout << " КУРСОВАЯ РАБОТА\n";

cout << " на тему\n";

cout << " РАЗРАБОТКА ПРОГРАММЫ ПЛАНИРОВАНИЯ БЮДЖЕТА ПРОЕКТА\n\n\n";

cout << "Выполнил студент группы: 010101 ЮХНЕВИЧ ВАДИМ СЕРГЕЕВИЧ\n\n";

cout << "Преподаватель: МЕЖЕННАЯ МАРИНА МИХАЙЛОВНА\n";

cout << "\n\n\n\n\n\n МИНСК 2021";

\_getch();

}

void System(vector <Accounts>& vector\_of\_accs, vector <Workers>& vector\_of\_workers)

//основной интерфейс

{

bool mainMenuFlag = 1;

do

{

string signedAccount;

int role;

switch (outputMainMenu())

{

case '1':

role = -1;

signedAccount = loginAccount(vector\_of\_accs, role);

\_getch();

system("cls");

if (role == 1)

adminInterface(vector\_of\_accs,vector\_of\_workers, signedAccount);

if (role == 0)

userInterface(vector\_of\_accs, vector\_of\_workers, signedAccount);

break;

case '2':

registeringAccount(vector\_of\_accs);

break;

case '0':

if (confirmAction("выйти"))

{

inputInfoAboutProgram();

mainMenuFlag = 0;

}

break;

}

} while (mainMenuFlag);

}

void adminInterface(vector <Accounts>& vector\_of\_accs, vector <Workers>& vector\_of\_workers, string login)

//админский интерфейс

{

bool adminMenuFlag = 1;

do

{

switch (outputAdminMenu(login))

{

case '1':

workingWithAccounts(vector\_of\_accs, login);

break;

case '2':

workingWithWorkersAdmin(vector\_of\_workers);

break;

case '3':

login =changeLogin(vector\_of\_accs, login);

break;

case '4':

changePassword(vector\_of\_accs, login);

break;

case '0':

if (confirmAction("выйти из аккаунта"))

adminMenuFlag = 0;

break;

}

system("cls");

} while (adminMenuFlag);

}

char outputAdminMenu(string login)

//вывод к админскому интерфейсу

{

cout << "------" << login << "(admin)------\n";

cout << "1| Работа с учетными записями\n";

cout << "2| Работа с данными\n\n";

cout << "3| Смена логина учетной записи\n";

cout << "4| Смена пароля учетной записи\n";

cout << "0| Выход\n";

return choiceAction('0', '4');

}

void workingWithAccounts(vector <Accounts>& vector\_of\_accs, string login)

//интерфей работы с аккаунтами для админа

{

bool menuFlag = 1;

do

{

outputAllAccounts(vector\_of\_accs);

switch (outputWorkingWithAccountsMenu())

{

case '1':

verificationAccount(vector\_of\_accs);

break;

case '2':

deleteUserAccount(vector\_of\_accs,login);

break;

case '3':

changeRoleAccount(vector\_of\_accs, login);

break;

case '0':

menuFlag = 0;

break;

}

system("cls");

} while (menuFlag);

}

char outputWorkingWithAccountsMenu()

//меню для того что сверху

{

cout << "1| Верификация учетных записей\n";

cout << "2| Удалениие учетных записей\n";

cout << "3| Изменение роли учетных записей\n";

cout << "0| Назад\n";

return choiceAction('0', '3');

}

void workingWithWorkersAdmin(vector <Workers>& vector\_of\_workers)

//интерфейс работы с данными работяг для админов

{

bool menuFlag = 1;

do

{

outputAllWorkers(vector\_of\_workers);

switch (outputWorkingWithWorkersMenu())

{

case '1':

addWorker(vector\_of\_workers);

break;

case '2':

editWorker(vector\_of\_workers);

break;

case '3':

deleteWorker(vector\_of\_workers);

break;

case '4':

individualTask(vector\_of\_workers);

break;

case '5':

sortByCompany(vector\_of\_workers);

break;

case '6':

sortByWork(vector\_of\_workers);

break;

case '7':

sortByHours(vector\_of\_workers);

break;

case '8':

sortByMoney(vector\_of\_workers);

break;

case '0':

menuFlag = 0;

break;

}

system("cls");

} while (menuFlag);

}

char outputWorkingWithWorkersMenu()

//меню к тому что сверху

{

cout << "1| Добавить запись\n";

cout << "2| Редактиовать запись\n";

cout << "3| Удалить запись\n";

cout << "4| Индивидуальное задание\n";

cout << "5| Отсортировть по названию проекта\n";

cout << "6| Отсортировть по типу работы\n";

cout << "7| Отсортировть по количеству часов\n";

cout << "8| Отсортировть по заработной плате\n";

cout << "0| Назад\n";

return choiceAction('0', '8');

}

void userInterface(vector <Accounts>& vector\_of\_accs, vector <Workers>& vector\_of\_workers, string login)

//интерфейс пользователя

{

bool userMenuFlag = 1;

do

{

switch (outputUserMenu(login))

{

case '1':

workingWithWorkersUser(vector\_of\_workers);

break;

case '2':

if (workingWithAccountByUser(vector\_of\_accs, login))

userMenuFlag = 0;

break;

case '0':

if (confirmAction("выйти из аккаунта"))

userMenuFlag = 0;

break;

}

system("cls");

} while (userMenuFlag);

}

char outputUserMenu(string login)

//меню к тому что сверху

{

cout << "--------" << login << "--------\n";

cout << "1| Работа с данными\n";

cout << "2| Работа со своей учетной записью\n";

cout << "0| Выход\n";

return choiceAction('0', '2');

}

bool workingWithAccountByUser(vector <Accounts>& vector\_of\_accs, string login)

//интерфейс юзера для работы с изменениями своего акка

//вернет 0-если все хорошо 1-если аккаунт удален и требуется выйти к основному меню

{

bool menuFlag = 1;

do

{

switch (outputWorkingWithUserAccountMenu())

{

case '1':

login = changeLogin(vector\_of\_accs, login);

break;

case '2':

changePassword(vector\_of\_accs, login);

break;

case '3':

if(deleteYourselfAccount(vector\_of\_accs, login))

return 1;

break;

case '0':

menuFlag = 0;

break;

}

system("cls");

} while (menuFlag);

return 0;

}

char outputWorkingWithUserAccountMenu()

{

cout << "1| Смена логина учетной записи\n";

cout << "2| Смена пароля учетной записи\n";

cout << "3| Удаление своей учентной записи\n";

cout << "0| Назад\n";

return choiceAction('0', '3');

}

void workingWithWorkersUser(vector <Workers>& vector\_of\_workers)

//интерфейс для работы с данными работяг для юзера

{

bool menuFlag = 1;

do

{

outputAllWorkers(vector\_of\_workers);

switch (outputWorkingWithWorkersMenuUser())

{

case '1':

individualTask(vector\_of\_workers);

break;

case '2':

sortByCompany(vector\_of\_workers);

break;

case '3':

sortByWork(vector\_of\_workers);

break;

case '4':

sortByHours(vector\_of\_workers);

break;

case '5':

sortByMoney(vector\_of\_workers);

break;

case '0':

menuFlag = 0;

break;

}

system("cls");

} while (menuFlag);

}

char outputWorkingWithWorkersMenuUser()

{

cout << "1| Индивидуальное задание\n";

cout << "2| Отсортировть по названию проекта\n";

cout << "3| Отсортировть по типу работы\n";

cout << "4| Отсортировть по количеству часов\n";

cout << "5| Отсортировть по заработной плате\n";

cout << "0| Назад\n";

return choiceAction('0', '5');

}

**validation.h**

#pragma once

#include <iostream>

#include <string>

#include <conio.h>

using namespace std;

bool checkString(string str);

int getInt(int bein, int end);

char choiceAction(char begin, char end);

bool confirmAction(string str);

string inputName();

string inputCompany();

**validation.cpp**

#include "validation.h"

bool checkString(string str)

//проверка строки на валидные символы 1-строка с ошибкой 0-строка верна

{

int size = str.length();

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

{

if (!((str[i] >= 'a' && str[i] <= 'z') ||

(str[i] >= 'A' && str[i] <= 'Z') ||

(str[i] >= '0' && str[i] <= '9') ||

str[i] == '\_' || str[i] == '.'

))//если недопустимые символы

{

cout << "Ошибка:недопустимые символы.(Разрешенные символы:[a-z][A-Z][0-9]и '\_' и '.')\n";

return 1;

}

if (!(size > 3 && size < 17))

{

cout << "Ошибка:допустимый размер 4-16 символов\n";

return 1;

}

}

return 0;//если прошло проверку

}

int getInt(int begin, int end)

//ф-я для ввода только целого значения из заданного(переданного) диапазона

{

int input;

while (true)

{

cin >> input;

if (cin.get() != '\n') // если данные некорректны

{

cout << "Некорректный ввод,введите корректное значение!\n-> ";

cin.clear(); // восстанавливаем поток

cin.ignore(32767, '\n'); // пропускаем всё до \n

}

else

{

if (!(input >= begin && input <= end))

{

cout << "Некорректный ввод!(допустимые значения [" << begin << ";" << end << "] Введите корректное значение!\n-> ";

continue; // возвращаемся к вводу ещё раз

}

break; // если данные корректны, останавливаем цикл

}

}

return input;

}

char choiceAction(char begin, char end)

//ф-я для считывания с клавиатуры только входящих в диапазон символов

{

char choice;

do

{

choice = \_getch();

} while (!(choice >= begin && choice <= end));

system("cls");

return choice;

}

bool confirmAction(string str)

//1-подвержден 0-отмена

//ф-я для подтверждения необратимого действия

{

cout << "Вы действительно хотите " + str + "?\n";

cout << "1| Да\n";

cout << "2| Нет\n";

char choice;

do

{

choice = \_getch();

} while (!(choice == '1' || choice == '2'));

system("cls");

if (choice == '1')

return 1;

else

return 0;

}

string inputName()

//ф-я для ввода валидных символов для ячеек имени и должности

{

bool flag;

string work;

do

{

flag = 0;

cin >> work;

for (int i = 0; i < work.size(); i++)

{

if (!((work[i] >= 'a' && work[i] <= 'z') ||

(work[i] >= 'A' && work[i] <= 'Z') ||

(work[i] >= 'а' && work[i] <= 'я') ||

(work[i] >= 'А' && work[i] <= 'Я') ||

work[i] == '(' || work[i] == ')' ||

work[i] == '\_'))

{

cout << "Ошибка:недопустимый символ.Повторите ввод.\n-> ";

flag = 1;

break;

}

}

} while (flag);

return work;

}

string inputCompany()

//ф-я для ввода любых символов исключая пробел

{

bool flag;

string company;

do

{

flag = 0;

cin >> company;

for (int i = 0; i < company.size(); i++)

{

if (company[i]==' ')

{

cout << "Ошибка:недопустимый символ(ПРОБЕЛ).Повторите ввод.\n-> ";

flag = 1;

break;

}

}

} while (flag);

return company;

}