Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра инженерной психологии и эргономики

Дисциплина: Программируемые мобильные системы

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

на тему

**РАЗРАБОТКА ПРОГРАММЫ УЧЕТА ЗАРАБОТНОЙ ПЛАТЫ СОТРУДНИКОВ ПРЕДПРИЯТИЯ**

Выполнил: студент группы

Ф.И.О.

Проверил: Ф.И.О. преподавателя

Минск 2022

**СОДЕРЖАНИЕ**

Оглавление

1. **Требования к программе1**

Введите название главы (уровень 2)2

Введите название главы (уровень 3)3

1. **Конструирование программы4**

Введите название главы (уровень 2)5

Введите название главы (уровень 3)6

1. Требования к программе.

2. Конструирование программы.

2.1 Разработка модульной структуры программы.

2.2 Выбор способа организации данных.

2.3 Разработка перечня пользовательских функций программы.

3. Разработка алгоритмов работы программы.

3.1 Алгоритм функции main.

3.2 Алгоритм функции enableRussianLanguage.

3.3 Алгоритм функции addUser.

4. Описание работы программы.

4.1 Авторизация.

4.2 Модуль администратора.

4.3 Модуль пользователя.

4.4 Исключительные ситуации.

Приложение А Листинг кода main.cpp.

Приложение Б Листинг кода input.h.

Приложение В Листинг кода crypt.h.

# 1 ТРЕБОВАНИЯ К ПРОГРАММЕ

Необходимо разработать программу расчета заработной платы сотрудников предприятия.

Сведения о сотрудниках предприятия содержат:

- Ф.И.О. сотрудника;

- табельный номер;

- год;

- месяц;

- количество проработанных часов за месяц;

- почасовой тариф.

Рабочее время свыше 144 часов считается сверхурочным и считается в двойном размере.

Индивидуальное задание: рассчитать размер заработной платы каждого сотрудника за вычетом подоходного налога, который составляет 12%. Определить объем выплат конкретному сотруднику за требуемый период времени, который вводится с клавиатуры.

Также необходимо реализовать авторизацию для входа в систему, функционал администратора и пользователя.

* 1. Исходные данные

1. Тема: разработка программы расчета заработной платы сотрудников предприятия
2. Язык программирования: C++
3. Среда разработки: Microsoft Visual Studio
4. Вид приложения: консольное
5. Парадигма программирования: процедурная
6. Способ организации данных: структуры
7. Способ хранения данных: файлы
8. Каждая логически завершенная подзадача должна быть реализована в виде отдельной функции
9. Построение программного кода должно соответствовать соглашению о коде «C++ Code Convention»
10. К защите курсовой работы представляются: консольное приложение и пояснительная записка
11. Текст пояснительной записки оформляется в соответствии со стандартом предприятия СТП 01-2017
    1. Функциональные требования

Первый этап работы программы – авторизация – предоставление прав доступа.

В рамках данного этапа считываются данные из файла с учетными записями пользователей следующего вида:

- login;

- hashPassword;

- role;

- access.

Поле login хранит логин пользователя, hashPassword хранит зашифрованный пароль пользователя, role хранит права доступа, где 1 – администратор, 0 – пользователь, access хранит разрешение на вход в систему, где 1 – доступ разрешен, 0 – доступ запрещен.

Если файла с учетными записями пользователей не существует, то создается файл, где записываются данные администратора.

По соображениям безопасности, при вводе пароля на экране выводятся «звездочки».

Регистрация новых пользователей осуществляется администратором в режиме работы с учетными записями пользователе. При регистрации выполняется проверка на уникальность логина.

Также возможна регистрация пользователей при входе, после которой администратор должен подтвердить регистрацию в режиме редактирования учетных записей пользователей, путем изменения поля access.

Второй этап работы программы – работа с данными, которая становится доступной только после успешного входа в систему. Данные о сотрудниках хранятся в файле и имеют структуру:

- уникальный ID записи;

- фамилия сотрудника;

- имя сотрудника;

- отчество сотрудника;

- табельный номер;

- год;

- месяц;

- количество проработанных часов за месяц;

- почасовой тариф.

Для работы с данными предусмотрены два функциональных модуля: модуль администратора и модуль пользователя.

Модуль администратора содержит следующие подмодули:

1. Управление учетными записями пользователей:

- просмотр всех учетных записей пользователей

- добавление новой учетной записи пользователя

- редактирование учетной записи пользователя

- удаление учетной записи пользователя

Так как существует возможность регистрации при входе, администратор подтверждает аккаунт через подмодуль «Редактирование учетной записи пользователя».

Допускается создавать учетную запись пользователя для нескольких администраторов.

Администратору запрещено удалять или изменять права доступа и разрешение на вход в систему самому себе.

1. Работа с данными:
   1. Режим редактирования:

- просмотр всех записей;

- добавление новой записи;

- удаление записи;

- редактирование записи;

* 1. Режим обработки данных:

- выполнение индивидуального задания в рамках данной курсовой работы;

- поиск данных;

- сортировка;

Модуль пользователя содержит следующие подмодули:

1. Просмотр всех данных;
2. Режим обработки данных:

- выполнение индивидуального задания в рамках данной курсовой работы;

- поиск данных;

- сортировка;

Для реализации перечисленных модулей и подмодулей созданы меню с соответствующими пунктами.

В данной курсовой работе предусмотрены:

1. Исключительные ситуации:

- отсутствие файла с учетными записями пользователей;

- проверка на уникальность логина при регистрации и редактировании учетной записи пользователя;

- запрет на изменение некоторых полей и удаление учетной записи пользователя;

- введенные пользователем данные не соответствуют формату поля;

- введенные пользователем данные нелогичны (например, почасовой тариф меньше либо равен 0);

- файл с данными для чтения не существует;

- номер удаляемой записи выходит за пределы массива;

1. Возможность перемещения по программе;
2. Запрос на удаление при необратимых действиях: подтверждение при удалении данных;
3. Обратную связь с пользователем через сообщения.

## **1.3 Требования к программной реализации**

1. Все переменные и константы должны иметь осмысленные имена в рамках тематики данной курсовой работы. Переменным рекомендуется присваивать имена, состоящие из букв нижнего регистра, для формирования составного имени используется «верблюжья нотация» (например, flagExit). Константам присваиваются имена, состоящие из букв верхнего регистра, и используется нижнее подчеркивание (например, FILE\_OF\_ACCOUNTS);
2. Имена функции должны быть осмысленными, начинаются с буквы нижнего регистра и строятся по принципу глагол+существительное (например, addAccount). Если функция выполняет проверку и возвращает результат типа bool, то ее название должно начинаться с глагола is (например, isNumberNumeric);
3. Не допускается использование оператора прерывания goto
4. Код не должен содержать неименованных числовых констант, неименованных строковых констант. 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 дальнейшая работа ведется с этим массивом, а не происходит многократное считывание данных из файла в каждой функции).
9. Следует избегать глубокой вложенности условных и циклических кон-струкций: вложенность блоков должна быть не более трех.
10. Следует избегать длинных функций: текст функции должен умещаться на один экран (размер текста не должен превышать 25–50 строк).
11. Следует выносить код логически независимых модулей в отдельные .cpp файлы и подключать их с помощью заголовочных .h файлов.

# **2 КОНСТРУИРОВАНИЕ**

## **2.1 разработка модульной структуры**

## **2.2 выбор способа организации данных**

В качестве способа организации данных используются структуры следующего вида:

- Структура учётной записи пользователя:

- логин;

- зашифрованный пароль;

- права доступа;

- возможность входа в систему;

- Структура записи сотрудника:

- уникальный ID записи;

- фамилия сотрудника;

- имя сотрудника;

- отчество сотрудника;

- табельный номер;

- год;

- месяц;

- количество проработанных часов за месяц;

- почасовой тариф.

В качестве способа объединения входных данных используются глобальные динамические массивы структур.

## **2.3 Разработка перечня пользовательских функций**

void enableRussianLanguage() – функция, которая подключает поддержку русского языка в консоли.

void login() – функция авторизации пользователей.

void exit() – функция завершения работы программы.

Блок функций, отвечающих за работу с файлом и динамическим массивом структур учетных записей пользователей:

- void cleanMemoryUsers() – функция, очищающая динамический массив структур учетных записей пользователей;

- void addAdmin\_fileWasntExist() – функция, добавляющая учетную запись администратора при создании файла учетных записей пользователей, если файл при запуске не существовал;

- void addUser() – функция, добавляющая новую пустую структуру в динамический массив структур учетных записей пользователей;

- void openUserFile() – функция, открывающая файл с учетными записями пользователями и передающая информацию из файла в динамический массив структур учетных записей пользователей;

- void createUserFile() – функция, создающая файл с учетными записями пользователей, если он отсутствует;

- void toUserFile() – функция, передающая информацию из динамического массива структур учетных записей пользователей в файл.

Блок функций-модулей пользователей разных прав доступа к системе:

- void adminMenu() – функция-модуль администратора;

- void userMenu() – функция-модуль пользователя.

Блок функций, отвечающих за работу с файлом и динамическим массивом структур сотрудников:

- void enterDataFilename() – функция, в которой пользователь вводит название файла с сотрудниками;

- void openDataFile() – функция, открывающая файл с сотрудниками и передающая информацию из файла в динамический массив структур сотрудников;

- void cleanMemoryData() – функция, очищающая динамический массив структур сотрудников;

- void createDataFile() – функция, создающая файл с сотрудниками, если он отсутствует;

- void addEmployee() – функция, добавляющая новую пустую структуру в динамический массив сотрудников;

- void toDataFile() – функция, передающая информацию из динамического массива структур сотрудников в файл.

Блок функций-подмодулей администратора:

- void userControlPanel() – функция-подмодуль управления учетными записями пользователей;

- void dataAdmin() – функция-подмодуль работы с данными о сотрудниках.

Блок функций-подмодуля управления учетными записями пользователей:

- void showAllUsers() – функция просмотра содержимого динамического массива структур учетных записей пользователей;

- void addUserInDatabase() – функция, добавляющая новую учетную запись пользователя в динамический массив структур;

- void editUser() – функция редактирования учетной записи пользователя в динамическом массиве структур;

- void deleteUser() – функция удаления учетной записи пользователя из динамического массива структур.

Блок функций-подмодуля управления учетными записями пользователй:

- void outputUsers() – функция вывода учетных записей пользователей с шапкой;

- void outputUser(int, bool) – функция вывода одной учётной записи пользователя;

- int getIndexOfUser() – функция, получающая индекс учетной записи пользователя в динамическом массиве структур;

- bool isAdminEditsThemself(int) – функция, которая проверяет: не редактирует ли администратор самого себя;

- void updateMaxLengthOfUserDatabase() – функция, обновляющая максимальную длину компонентов структуры.

Блок функций-подмодуля работы с данными о сотрудниках:

- void modifyData() – функция-подмодуль режима редактирования данных;

- void dataProcessing() – функция-подмодуль режима обработки данных.

Блок функций-подмодуля режима обработки данных:

- void salary() – функция вывода зарплаты сотрудников;

- void linearSearch() – функция линейного поиска данных сотрудников;

- void sort() – функция сортировки данных сотрудников.

Блок функций-подмодуля режима редактирования данных о сотрудниках:

- void viewData() – функция просмотра содержимого динамического массива структур сотрудников;

- void addData() – функция добавления записи в динамический массив структур сотрудников;

- void deleteData() – функция удаления записи из динамического массива структур сотрудников;

- void editData() – функция редактирования записи в динамическом массиве структур сотрудников.

Блок функций-подмодулей режима редактирования данных о сотрудниках:

- void outputData() – функция вывода всех записей;

- void outputOneData(int) – функция вывода одной записи;

- int getLocalYear() – функция получения года в системе;

- string toLower(string) – функция, опускающая регистр всего слова;

- int getFreeTabelNumber() – функция получения свободного табельного номера;

- int getFreeID() – функция получения свободного номера.

Блок функций-подмодулей режима обработки данных о сотрудниках:

- double getSalary(int, double) – функция вычисления зарплаты по количеству часов и почасовому тарифу;

- void sort\_getParametr(bool) – функция выбора критерия сортировки;

- void showCriteria() – функция вывода критериев сортировки;

- void bubbleSort(int) – функция пузырьковой сортировки;

- void quicksort(int, int, int) – функция быстрой сортировки;

- int getMiddleIndex(int, int, int) – вспомогательная функция быстрой сортировки;

- string getTempParametr(employee, int) – функция получения параметра из структуры сотрудника для поиска;

- int getID() – функция получения номера записи в динамическом массиве структур сотрудников;

- int getMaxID() – функция получения максимального номера записи в динамическом массиве структур сотрудников.

# **3 РАЗРАБОТКА АЛГОРИТМОВ РАБОТЫ ПРОГРАММЫ**

## **3.1 Алгоритм функции main**



## **3.2** **Алгоритм функции enableRussianLanguage**



## **3.3 Алгоритм функции addUser**



# **4 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ**

Описание работы программы подразумевает краткое словесное описание работы программы со скриншотами консоли

## **4.1 Авторизация**

Когда запускается функция авторизации, запускается открытие файла с учетными записями пользователей в динамический массив структур. Затем выводится сообщение-предложение с вводом логина. Затем запускается бесконечный цикл, где пользователем вводится пароль. После ввода пароля, запускается цикл с нахождением совпадения логина в динамическом массиве структур учетных записей пользователей. Если программа находит совпадение, она сравнивает пароль, введённый пользователем, с паролем аккаунта из системы. Если совпадение есть, то программа выходит из бесконечного цикла, если совпадения нет, то выводится сообщение о неправильном пароле и пользователь снова вводит пароль. После бесконечного цикла, программа проверяет наличие аккаунта в динамическом массиве структур учетных записей пользователей, если учетная запись, а именно её индекс в динамическом массиве структур учетных записей пользователей не должен превышать общее количество учетных записей, существует в динамическом массиве, то в глобальные переменные присваивается логин и права доступа этой учетной записи, затем проверяется разрешение на доступ в систему, если доступ есть, то в зависимости от прав доступа запускается либо модуль администратора, либо модуль пользователя, если разрешение отсутствует, то выводится сообщение о закрытом доступе и предложение с входом в другой аккаунт. Если учетной записи пользователя не существует, а именно индекс учетной записи превышает количество учетных записей в динамическом массиве учетных записей пользователей, то выводится предложение с регистрацией аккаунта. Если пользователь желает зарегистрироваться, то создаётся новая пустая структура, в которую переносится логин, пароль, и права доступа, которые пользователь желает иметь. После выбора, файл с учетными записями обновляется. Если пользователь не желает зарегистрироваться, то программа завершает свою работу.

## **4.2 Модуль администратора**

При запуске модуля администратора, запускается бесконечный цикл – меню, в котором пользователь может выбрать запуск одного из двух подмодулей, выход из системы или завершение работы. При выборе одного из модуля, из файла считываются данные в динамический массив структур, логическая переменная editsInStructures, которая отвечает за изменения в динамическом массиве структур, и запускается подмодуль, который выбрал пользователь. Если пользователь желает выйти из системы, запускается функция авторизации. Если пользователь желает завершить работу программы, запускается функции завершения работы программы. Если пользователь ввёл неправильный номер, то выводится сообщение о неправильном номере и пользователю снова предлагается выбрать пункт меню. Если пользователь ввёл значение, отличное от int, то данное исключение обрабатывается в функции inputInt библиотеки input.h.

## **4.3 Модуль пользователя**

Модуль пользователя работает аналогично, как и модуль администратора, но отличается другим набором меню: просмотр и обработка данных их файла с сотрудниками, открытие другого файла с сотрудниками, выход из системы и завершение работы.

## **4.4 Исключительные ситуации**

В данной курсовой работы предусмотрены различные исключительные ситуации:

- отсутствие файла с учётными записями пользователей;

- проверка на уникальность логина при регистрации и редактировании учетной записи пользователя;

- если администратор вошёл в систему, то ему запрещается изменять себе некоторые поля или удалять свою учетную запись;

- введенные пользователем данные не соответствуют формату поля;

- введенные пользователем данные нелогичны (например, почасовой тариф меньше либо равен 0);

- файл с данными для чтения не существует;

- номер удаляемой записи выходит за пределы массива.

# **ПРИЛОЖЕНИЕ А**

#include <cmath>

#include <ctime>

#include <fstream>

#include <iostream>

#include <string>

#include <Windows.h>

#include "crypto.h"

#include "input.h"

#pragma warning (disable: 4996)

using namespace std;

struct user {

string login; // строка для хранения логина пользователя

string hashPassword; // строка для хранения шифрованного пароля пользователя

bool role; // логическая переменная, хранящая права пользователя (1 - admin, 0 - user)

bool access; // логическая переменная, хранящая возможность доступа в систему (1 - admin allows access to the system, 0 - admin doesn't allow access to the system)

};

int countOfUsers = 0; // целочисленная переменная, хранящая количество пользователей в системе

user\* Users = NULL; // указатель типа user, который используется как динамический массив структур, в которой храним информацию об аккауте

const char FILENAME\_OF\_USERS\_DATABASE[] = "users.txt"; // константный массив, хранящий название файла базы данных акканутов пользователей

bool editsInStructures = false; // логическая переменная, которая используется, чтобы проверить, что учётные записи в файле и учётные записи в массив структур отличаются

int maxLengthInUsersDatabase[3] = { 5, 6, 15 }; // 0 - login, 1 - password, 2 - role

string loginOfUser; // строка, которая хранит логин пользователя, который вошёл в систему

bool roleOfUser; // логическая переменная, которая хранит права пользователя, который вошёл в систему

struct employee {

int id;

string name;

string surname;

string middleName;

int tabelNumber = -1;

int year;

int month;

double countOfHours;

double rate;

};

employee\* Employees;

int countOfEmployees;

int sizeOfName = 0;

char \*fileName;

int maxLengthInDataDatabase[7] = { 3, 7, 8, 15, 3, 5, 30 }; // 0 - name, 1 - surname, 2 - middleName, 3 - tabelNumber, 4 - year, 5 - month, 6 - countOfHours

void enableRussianLanguage(); // функция, подключающая поддержку русского языка в консоли

void login(); // функция авторизации в систему

void exit(); // функция завершения работы с системой

// блок функций, отвечающий за работу с файлом и динамическим массивом структур учетных записей пользователей

void cleanMemoryUsers(); // функция, очищающая динамический массив структур учетных записей пользователей

void addAdmin\_fileWasntExist(); // функция, добавляющая учетную запись администратора при создании файла users.txt, если файл при запуске не существовал

void addUser(); // функция, добавляющая новую пустую структуру в динамический массив структур

void openUserFile(); // функция, открывающая файл с учетными записями пользователями и пере-дающая информацию из файла в динамический массив структур учетных записей пользователей

void createUserFile(); // функция, создающая файл с учетными записями пользователей, если он отсутствует

void toUserFile(); // функция, передающая информацию из динамического массива структур учетных записей пользователей в файл

// блок функций-модулей пользователей разных прав доступа к системе

void adminMenu(); // функция-модуль администратора

void userMenu(); // функция-модуль пользователя

// блок функций, отвечающих за работу с файлом и динамическим массивом структур сотрудников

void enterDataFilename(); // функция, в которой пользователь вводит название файла с сотрудниками

void openDataFile(); // функция, открывающая файл с сотрудниками и передающая информацию из файла в динамический массив структур сотрудников

void cleanMemoryData(); // функция, очищающая динамический массив структур сотрудников

void createDataFile(); // функция, создающая файл с сотрудниками, если он отсутствует

void addEmployee(); // функция, добавляющая новую пустую структуру в динамический массив сотрудников

void toDataFile(); // функция, передающая информацию из динамического массива структур сотрудников в файл

// блок функций-подмодулей администратора

void userControlPanel(); // функция-подмодуль управления учетными записями пользователей

void dataAdmin(); // функция-подмодуль работы с данными о сотрудниках

// блок функций-подмодуля управления учетными записями пользователей

void showAllUsers(); // функция просмотра содержимого динамического массива структур учетных записей пользователей

void addUserInDatabase(); // функция, добавляющая новую учетную запись пользователя в динамический массив структур

void editUser(); // функция редактирования учетной записи пользователя в динамическом массиве структур

void deleteUser(); // функция удаления учетной записи пользователя из динамического массива структур

// блок функций-подмодулей управления учетными записями

void outputUsers(); // функция вывода учетных записей пользователей с шапкой

void outputUser(int, bool); // функция вывода одной учётной записи пользователя

int getIndexOfUser(); // функция, получающая индекс учетной записи пользователя в динамическом массиве структур

bool isAdminEditsThemself(int); // функция, которая проверяет: не редактирует ли администратор самого себя

void updateMaxLengthOfUserDatabase(); // функция, обновляющая максимальную длину компонентов структуры

// блок функций-подмодуля работы с данными о сотрудниках

void modifyData(); // функция-подмодуль режима редактирования данных

void dataProcessing(); // функция-подмодуль режима обработки данных

// блок функций-подмодуля режима обработки данных

void salary(); // функция вывода зарплаты сотрудников

void linearSearch(); // функция линейного поиска данных сотрудников

void sort(); // функция сортировки данных сотрудников

// блок функций-подмодуля режима редактирования данных о сотрудниках

void viewData(); // функция просмотра содержимого динамического массива структур сотрудников

void addData(); // функция добавления записи в динамический массив структур сотрудников

void deleteData(); // функция удаления записи из динамического массива структур сотрудников

void editData(); // функция редактирования записи в динамическом массиве структур сотрудников

// блок функций-подмодулей режима редактирования данных о сотрудниках

void outputData(); // функция вывода всех записей

void outputOneData(int); // функция вывода одной записи

int getLocalYear(); // функция получения года в системе

string tolower(string); // функция, опускающая регистр всего слова

int getFreeTabelNumber(); // функция получения свободного табельного номера

int getFreeID(); // функция получения свободного номера

// блок функций-подмодулей режим обработки данных о сотрудниках

double getSalary(int, double); // функция вычисления зарплаты по количеству часов и почасовому тарифу

void sort\_getParametr(bool); // функция выбора критерия сортировки

void showCriteria(); // функция вывода критериев сортировки

void bubbleSort(int); // функция пузырьковой сортировки

void quickSort(int, int, int); // функция быстрой сортировки

int getMiddleIndex(int, int, int); // вспомогательная функция быстрой сортировки

string getTempParametr(employee, int); // функция получения параметра из структуры сотрудника для поиска

int getID(); // функция получения номера записи в динамическом массиве структур сотрудников

int getMaxID(); // функция получения максимального номера записи в динамическом массиве структур сотрудников

int main() {

enableRussianLanguage();

login();

}

void enableRussianLanguage() {

SetConsoleOutputCP(1251); // включение поддержки русского языка на вывод из консоли

SetConsoleCP(1251); // включение поддержки русского языка на ввод в консоль

}

void login() {

openUserFile();

system("cls");

cout << "Введите логин: ";

string loginString = inputLogin(); // строка логина, введённого с клавиатуры

string password; // строка пароля, введённого с клавиатуры

int i; // целочисленная переменная для цикла и определённого аккаунта из массива структур

bool find; // переменная, которая контролирует поиск аккаунта в массиве структур

while (true) {

cout << endl << "Введите пароль: ";

password = inputPassword\_hide();

find = true;

for (i = 0; i < countOfUsers; i++) {

if (loginString == Users[i].login)

if (password == decrypt(Users[i].hashPassword))

break;

else {

cout << endl << "Введён неправильный пароль, попробуйте ещё раз";

find = false;

continue;

}

}

if (find)

break;

}

if (find)

if (i < countOfUsers) {

loginOfUser = Users[i].login;

roleOfUser = Users[i].role;

if (Users[i].access)

roleOfUser ? adminMenu() : userMenu();

else {

cout << endl << "Доступ запрещён" << endl << "Хотите зайти в систему под другим логином? (0 - нет, 1 - да): ";

inputBool() ? login() : exit();

}

}

else {

cout << endl << "Чтобы зарегестрироваться, необходимо разрешение администратора, хотите оставить заявку? (0 - нет, 1 - да): ";

if (inputBool()) {

addUser();

Users[countOfUsers - 1].login = loginString;

Users[countOfUsers - 1].hashPassword = crypt(password);

cout << endl << "Какие права хотите получить? (1 - администратор, 0 - пользователь): ";

Users[countOfUsers - 1].role = inputBool();

Users[countOfUsers - 1].access = false;

toUserFile();

}

else

exit();

}

}

void exit() {

cleanMemoryUsers();

cleanMemoryData();

exit(0);

}

void cleanMemoryUsers() {

if (countOfUsers != 0 || Users != NULL) { // если в системе что-то есть

countOfUsers = 0;

delete [] Users;

Users = NULL;

}

}

void addAdmin\_fileWasntExist() {

Users = new user[++countOfUsers]; // создаём структуру размера countOfUsers + 1

cout << "Введите логин администратора: ";

Users[0].login = inputLogin();

cout << endl << "Введите пароль администратора: ";

Users[0].hashPassword = crypt(inputPassword());

Users[0].role = true;

Users[0].access = true;

}

void addUser() {

user\* temp = new user[++countOfUsers]; // временный массив структур аккаунтов пользователей

for (int i = 0; i < countOfUsers - 1; i++)

temp[i] = Users[i];

swap(temp, Users);

delete[] temp;

}

void openUserFile() {

cleanMemoryUsers();

ifstream readFile(FILENAME\_OF\_USERS\_DATABASE);

if (!readFile.is\_open()) {

createUserFile();

addAdmin\_fileWasntExist();

}

else {

char line[500]; // символьный массив для получения информации из файла

while (!readFile.eof()) {

readFile.getline(line, 500);

int count = 0; // целочисленная переменная для хода по структуре

addUser();

string temp; // временная строка для будущих компонентов структуры

for (int i = 0; i < 500 || line[i] != '\0'; i++) {

if (line[i] == ' ' || line[i] == ';') {

if (count < 2)

maxLengthInUsersDatabase[count] = maxLengthInUsersDatabase[count] < temp.length() ? temp.length() : maxLengthInUsersDatabase[count];

switch (count++) {

case 0:

Users[countOfUsers - 1].login = temp;

break;

case 1:

Users[countOfUsers - 1].hashPassword = temp;

break;

case 2:

Users[countOfUsers - 1].role = stoi(temp); // stoi - перевод string в int, но так как bool хранит значение 1 или 0, то stoi можно использовать для перевода из string в bool

break;

case 3:

Users[countOfUsers - 1].access = stoi(temp);

}

temp.clear();

continue;

}

temp.push\_back(line[i]);

}

}

}

readFile.close();

updateMaxLengthOfUserDatabase();

}

void createUserFile() {

ofstream createFile(FILENAME\_OF\_USERS\_DATABASE); // при стандартном открытии файла в режиме записи, если файл отсутствует, то он создаётся

createFile.close();

ifstream checkIfCreatedFileExists(FILENAME\_OF\_USERS\_DATABASE);

cout << "Файл " << FILENAME\_OF\_USERS\_DATABASE << (checkIfCreatedFileExists.is\_open() ? " успешно создан" : " не был создан") << endl; // проверка и вывод сообщения об удачном или нет создании файла

}

void toUserFile() {

ofstream writeFile(FILENAME\_OF\_USERS\_DATABASE);

for (int i = 0; i < countOfUsers; i++) {

writeFile << Users[i].login << " " << Users[i].hashPassword << " " << Users[i].role << " " << Users[i].access << ";";

if (i != countOfUsers - 1)

writeFile << endl;

}

writeFile.close();

}

void adminMenu() {

while (true) {

system("cls");

cout << "Меню администратора" << endl;

cout << "1 - Управление учётными записями" << endl;

cout << "2 - Режим работы с данными" << endl;

cout << "3 - Выход из системы" << endl;

cout << "0 - Завершение работы" << endl;

cout << "Ваш выбор: ";

int menu = inputInt();

system("cls");

switch (menu) {

case 1: {

openUserFile();

editsInStructures = false;

userControlPanel();

}

break;

case 2: {

openDataFile();

editsInStructures = false;

dataAdmin();

}

break;

case 3:

login();

break;

case 0:

exit();

break;

default:

cout << "Выбран неправильный номер, попробуйте ещё раз" << endl;

}

}

}

void userMenu() {

if (countOfEmployees == 0)

openDataFile();

while (true) {

system("cls");

cout << "Меню пользователя" << endl;

cout << "1 - Просмотр всех данных" << endl;

cout << "2 - Режим обработки данных" << endl;

cout << "3 - Открыть другой файл" << endl;

cout << "4 - Выход из системы" << endl;

cout << "5 - Завершение работы" << endl;

cout << "Ваш выбор: ";

switch (inputInt()) {

case 1:

viewData();

break;

case 2:

dataProcessing();

break;

case 3: {

cleanMemoryData();

openDataFile();

}

break;

case 4: {

cleanMemoryData();

login();

}

break;

case 5:

exit();

default:

cout << endl << "Вы ввели неправильный номер, попробуйте ещё раз";

}

}

}

void enterDataFilename() {

system("cls");

cout << "Введите название файла: ";

string temp;

temp.append(input() + ".txt");

sizeOfName = temp.length();

fileName = new char[sizeOfName];

strcpy(fileName, temp.c\_str());

}

void openDataFile() {

enterDataFilename();

ifstream openFile(fileName);

system("cls");

if (!openFile.is\_open()) {

cout << "Файла с информацией о сотрудниках не существует" << endl << "Хотите создать? (1 - да, 0 - нет): ";

inputBool() ? createDataFile() : roleOfUser ? adminMenu() : userMenu();

}

cleanMemoryData();

char line[500];

while (!openFile.eof()) {

openFile.getline(line, 500);

int count = 0;

addEmployee();

string temp;

for (int i = 0; i < 500 || line[i] != '\0'; i++) {

if (line[i] == '|' || line[i] == ';') {

temp = count > 3 && temp == "-" ? "0" : temp;

if (count < 7)

maxLengthInDataDatabase[count] = maxLengthInDataDatabase[count] < temp.length() ? temp.length() : maxLengthInDataDatabase[count];

switch (count++) {

case 0:

Employees[countOfEmployees - 1].id = stoi(temp);

break;

case 1:

Employees[countOfEmployees - 1].name = temp;

break;

case 2:

Employees[countOfEmployees - 1].surname = temp;

break;

case 3:

Employees[countOfEmployees - 1].middleName = temp;

break;

case 4:

Employees[countOfEmployees - 1].tabelNumber = stoi(temp);

break;

case 5:

Employees[countOfEmployees - 1].year = stoi(temp);

break;

case 6:

Employees[countOfEmployees - 1].month = stoi(temp);

break;

case 7:

Employees[countOfEmployees - 1].countOfHours = stod(temp);

break;

case 8:

Employees[countOfEmployees - 1].rate = stod(temp);

}

temp.clear();

continue;

}

temp.push\_back(line[i]);

}

}

openFile.close();

updateMaxLengthOfDataDatabase();

cout << "Файл успешно открыт" << endl;

Sleep(5000);

}

void cleanMemoryData() {

if (countOfEmployees != 0 || Employees != NULL) {

delete[] Employees;

Employees = NULL;

countOfEmployees = 0;

}

}

void createDataFile() {

ofstream openFile(fileName);

system("cls");

cout << (openFile.is\_open() ? "Файл БД успешно создан" : "Файл БД не создан") << endl;

openFile.close();

return;

}

void addEmployee() {

employee\* temp = new employee[++countOfEmployees];

for (int i = 0; i < countOfEmployees - 1; i++)

temp[i] = Employees[i];

swap(temp, Employees);

delete[] temp;

}

void toDataFile() {

ofstream openFile(fileName);

for (int i = 0; i < countOfEmployees; i++) {

openFile << Employees[i].id << "|" << Employees[i].name << "|" << Employees[i].surname << "|" << Employees[i].middleName << "|" << Employees[i].tabelNumber << "|" << Employees[i].year << "|" << Employees[i].month << "|" << Employees[i].countOfHours << "|" << Employees[i].rate << ";";

if (i != countOfEmployees - 1)

openFile << endl;

}

openFile.close();

}

void userControlPanel() {

while (true) {

system("cls");

cout << "Меню управления учётными записями" << endl;

cout << "1 - Просмотр всех учётных записей" << endl;

cout << "2 - Добавление новой учётной записи" << endl;

cout << "3 - Редактирование учётной записи" << endl;

cout << "4 - Удаление учётной записи" << endl;

cout << "0 - Выйти в главное меню" << endl;

cout << "Ваш выбор: ";

int menu = inputInt();

system("cls");

switch (menu) {

case 1:

showAllUsers();

break;

case 2:

addUserInDatabase();

break;

case 3:

editUser();

break;

case 4:

deleteUser();

break;

case 0: {

if (editsInStructures) {

cout << "Вы внесли изменения в файл базу данных пользователей, хотите обновить файл? (0 - нет, 1 - да): ";

bool temp = inputBool();

if (temp)

toUserFile();

cleanMemoryUsers();

editsInStructures = false;

}

return;

}

default:

cout << "Выбран неправильный номер, попробуйте ещё раз" << endl;

}

}

}

void dataAdmin() {

while (true) {

system("cls");

cout << "Режим работы с данными" << endl;

cout << "1 - Режим редактирования" << endl;

cout << "2 - Режим обработки данных" << endl;

cout << "0 - Выйти в главное меню" << endl;

cout << "Ваш выбор: ";

int menu = inputInt();

switch (menu) {

case 1:

modifyData();

break;

case 2:

dataProcessing();

break;

case 3:

openDataFile();

break;

case 0: {

if (editsInStructures) {

cout << "Вы внесли изменения в файл базу данных сотрудников, хотите обновить файл? (0 - нет, 1 - да): ";

if (inputBool())

toDataFile();

}

cleanMemoryData();

editsInStructures = false;

adminMenu();

}

default:

cout << "Выбран неправильный номер, попробуйте ещё раз" << endl;

}

}

}

void showAllUsers() {

system("cls");

cout << "Режим просмотра: ";

outputUsers();

cout << endl << "Нажмите Esc, чтобы выйти в меню управления учётными записями";

while (true)

if (\_getch() == 27)

return;

}

void addUserInDatabase() {

system("cls");

addUser();

cout << "Добавление пользователя" << endl;

string login;

while (true) {

cout << "Введите логин: ";

login = inputLogin();

bool find = false;

for (int i = 0; i < countOfUsers - 1; i++)

if (login == Users[i].login) {

cout << endl << "Введите уникальный логин" << endl;

find = true;

break;

}

if (!find)

break;

}

Users[countOfUsers - 1].login = login;

cout << endl << "Введите пароль: ";

Users[countOfUsers - 1].hashPassword = crypt(inputPassword());

cout << endl << "Выберите уровень доступа (1 - admin, 0 - user): ";

Users[countOfUsers - 1].role = inputBool();

Users[countOfUsers - 1].access = true;

editsInStructures = true;

cout << endl << "Пользователь успешно добавлен в систему";

Sleep(5000);

updateMaxLengthOfUserDatabase();

editsInStructures = true;

return;

}

void editUser() {

system("cls");

cout << "Редактирование учётной записи" << endl;

outputUsers();

int index = getIndexOfUser();

cout << endl << "Показывать пароли? (1 - да, 0 - нет): ";

bool showPasswords = inputBool();

while (true) {

cout << endl << "Выбранный пользователь:" << endl;

outputUser(index, showPasswords);

cout << "Выберите параметр, который хотите изменить:" << endl << "1 - Логин" << endl << "2 - Пароль" << endl << "3 - Уровень доступа" << endl << "4 - Доступ к системе" << endl << "5 - Выбрать другую учётную запись" << endl << "6 - Выход в предыдущее меню" << endl << "Ваш выбор: ";

int menu = inputInt();

switch (menu) {

case 1: {

while (true) {

cout << endl << "Введите логин: ";

string login = inputLogin();

bool find = false;

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

if (i == index)

continue;

else

if (login == Users[i].login)

find = true;

if (!find) {

Users[index].login = login;

break;

}

else

cout << endl << "Данный логин уже используется, попробуйте другой";

}

}

break;

case 2: {

cout << endl << "Введите пароль: ";

Users[index].hashPassword = crypt(inputPassword());

}

break;

case 3: {

if (isAdminEditsThemself(index))

break;

cout << endl << "Введите уровень доступа (1 - admin, 0 - user): ";

Users[index].access = inputBool();

}

break;

case 4: {

if (isAdminEditsThemself(index))

break;

cout << endl << "Введите доступ к системе (1 - доступ разрешён, 0 - доступ запрещён): ";

Users[index].access = inputBool();

}

break;

case 5:

editUser();

break;

case 6:

return;

default:

cout << endl << "Вы ввели неправильный номер, попробуйте ещё раз" << endl;

}

if (menu > 0 && menu < 5) {

cout << endl << "Пользователь успешно изменён";

editsInStructures = true;

updateMaxLengthOfUserDatabase();

Sleep(5000);

}

}

}

void deleteUser() {

system("cls");

cout << "Удаление пользователя" << endl;

outputUsers();

int index = getIndexOfUser();

editsInStructures = true;

countOfUsers--;

user\* temp = new user[countOfUsers];

int j = 0;

for (int i = 0; i <= countOfUsers; i++) {

if (i == index)

continue;

temp[j++] = Users[i];

}

swap(temp, Users);

delete[] temp;

cout << endl << "Удаление завершено" << endl;

updateMaxLengthOfUserDatabase();

Sleep(5000);

editsInStructures = true;

return;

}

void outputUsers() {

cout << endl << "Показать пароли? (1 - да, 0 - нет): ";

bool showPasswords = inputBool();

printf("%\*s%\*s%\*s%\*s%\*s%\*s%\*s", to\_string(countOfUsers).length(), "#", 3, " | ", maxLengthInUsersDatabase[0], "Login", 3, " | ", maxLengthInUsersDatabase[1], "Password", 3, " | ", maxLengthInUsersDatabase[2], "Уровень доступа");

cout << " | Доступ к системе" << endl;

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

outputUser(i, showPasswords);

}

void outputUser(int index, bool showPasswords) {

printf("%\*i", to\_string(countOfUsers).length(), index + 1);

cout << " | ";

printf("%\*s", maxLengthInUsersDatabase[0], Users[index].login.c\_str());

cout << " | ";

string password;

if (showPasswords)

password = decrypt(Users[index].hashPassword);

else

for (int i = 0; i < Users[index].hashPassword.length(); i++)

password.push\_back('\*');

printf("%\*s", maxLengthInUsersDatabase[1], password.c\_str());

cout << " | ";

printf("%\*s", maxLengthInUsersDatabase[2], Users[index].role ? "администратор" : "пользователь");

cout << " | " << (Users[index].access ? "+" : "-") << endl;

}

int getIndexOfUser() {

cout << "Выберите пользователя: ";

while (true) {

int index = inputInt();

--index;

if (index >= 0 && index < countOfUsers)

return index;

else

cout << endl << "Введён неправильный номер пользователя, попробуйте ещё раз" << endl;

}

}

bool isAdminEditsThemself(int index) {

bool role = false;

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

if (i == index && loginOfUser == Users[i].login && Users[i].role)

role = true;

if (role)

cout << endl << "Вам запрещено запрещать себе доступ к системе" << endl;

return role;

}

void updateMaxLengthOfUserDatabase() {

maxLengthInUsersDatabase[0] = 5;

maxLengthInUsersDatabase[1] = 6;

for (int i = 0; i < 2; i++)

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

switch (i) {

case 0:

maxLengthInUsersDatabase[i] = maxLengthInUsersDatabase[i] < Users[j].login.length() ? Users[j].login.length() : maxLengthInUsersDatabase[i];

break;

case 1:

maxLengthInUsersDatabase[i] = maxLengthInUsersDatabase[i] < Users[j].hashPassword.length() ? Users[j].hashPassword.length() : maxLengthInUsersDatabase[i];

}

}

void dataProcessing() {

while (true) {

system("cls");

cout << "Режим обработки" << endl;

cout << "1 - Расчёт заработной платы" << endl;

cout << "2 - Поиск данных" << endl;

cout << "3 - Сортировка данных" << endl;

cout << "0 - Выход в предыдущее меню" << endl;

cout << "Ваш выбор: ";

switch (inputInt()) {

case 1:

salary();

break;

case 2:

linearSearch();

break;

case 3:

sort();

break;

case 0:

roleOfUser ? dataAdmin() : userMenu();

}

}

}

void modifyData() {

system("cls");

cout << "Режим редактированния данных" << endl;

cout << "1 - Просмотр всех данных" << endl;

cout << "2 - Добавления новой записи" << endl;

cout << "3 - Удаление записи" << endl;

cout << "4 - Редактирование записи" << endl;

cout << "0 - Вернуться в предыдущее меню" << endl;

cout << "Ваш выбор: ";

while (true) {

int menu = inputInt();

switch (menu) {

case 1:

viewData();

break;

case 2:

addData();

break;

case 3:

deleteData();

break;

case 4:

editData();

break;

case 0:

dataAdmin();

default:

cout << endl << "Выбран неправильный номер, попробуйте ещё раз" << endl;

}

}

}

void salary() {

while (true) {

while (true) {

system("cls");

cout << "Расчёт заработной платы" << endl;

outputData();

int id = getID(), i;

for (i = 0; i < countOfEmployees; i++)

if (id == Employees[i].id)

break;

while (true) {

system("cls");

cout << endl << "Выбранный сотрудник: " << endl;

printf("%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s", to\_string(countOfEmployees).length(), "#", 3, " | ", maxLengthInDataDatabase[0], "Имя", 3, " | ", maxLengthInDataDatabase[1], "Фамилия", 3, " | ", maxLengthInDataDatabase[2], "Отчество", 3, " | ", maxLengthInDataDatabase[3], "Табельный номер", 3, " | ", maxLengthInDataDatabase[4], "Год", 3, " | ", maxLengthInDataDatabase[5], "Месяц", 3, " | ", maxLengthInDataDatabase[6], "Количество проработанных часов");

cout << " | Почасовой тариф" << endl;

outputOneData(i);

cout << "Рассчитать зарплату за период или по информации записи (0 - за период, 1 - по информации записи): ";

bool temp = inputBool();

if (temp)

cout << endl << "Зарплата данного сотрудника: " << getSalary(i, Employees[i].countOfHours) << endl;

else {

double tempDouble;

while (true) {

cout << endl << "Введите требуемый период: ";

tempDouble = inputDouble();

if (tempDouble >= 0)

break;

else

cout << endl << "Вы ввели неправильное количество проработанных часов, попробуйте ещё раз";

}

cout << endl << "Зарплата данного сотрудника за " << tempDouble << " часов: " << getSalary(i, tempDouble) << endl;

}

cout << "Хотите выйти в предыдущее меню или выбрать другого сотрудника? (0 - предыдущее меню, 1 - выбрать другого сотрудника): ";

if (inputBool())

break;

else

dataProcessing();

}

}

}

}

void linearSearch() {

int criteria;

while (true) {

system("cls");

cout << "Линейный поиск" << endl;

showCriteria();

cout << "0 - Выход в предыдущее меню" << endl << "Ваш выбор: ";

switch (criteria = inputInt()) {

case 1:

case 2:

case 3:

case 4:

case 5:

case 6:

break;

case 0:

dataProcessing();

default:

cout << endl << "Вы ввели неправильный номер, попробуйте ещё раз";

}

if (criteria > 0 && criteria < 7)

break;

}

string criteriaString;

cout << "Введите параметр поиска: ";

if (criteria > 0 && criteria < 5)

criteriaString = to\_string(inputInt());

else {

criteriaString = to\_string(inputDouble());

if (stod(criteriaString) - stoi(criteriaString) == 0)

criteriaString = to\_string(stoi(criteriaString));

}

bool find = false;

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

if (criteriaString == getTempParametr(Employees[i], criteria)) {

if (!find) {

printf("%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s", to\_string(countOfEmployees).length(), "#", 3, " | ", maxLengthInDataDatabase[0], "Имя", 3, " | ", maxLengthInDataDatabase[1], "Фамилия", 3, " | ", maxLengthInDataDatabase[2], "Отчество", 3, " | ", maxLengthInDataDatabase[3], "Табельный номер", 3, " | ", maxLengthInDataDatabase[4], "Год", 3, " | ", maxLengthInDataDatabase[5], "Месяц", 3, " | ", maxLengthInDataDatabase[6], "Количество проработанных часов");

cout << " | Почасовой тариф" << endl;

}

find = true;

outputOneData(i);

}

if (!find)

cout << endl << "Совпадений не найдено" << endl;

cout << "Хотите продолжить поиск или выйти в предыдущее меню? (0 - продолжить поиск, 1 - выход в предыдущее меню): ";

inputBool() ? linearSearch() : dataProcessing();

}

void sort() {

while (true) {

system("cls");

cout << "Сортировка данных" << endl;

cout << "1 - Пузырьковая сортировка" << endl;

cout << "2 - Быстрая сортировка" << endl;

cout << "0 - Выход в предыдущее меню" << endl;

cout << "Ваш выбор: ";

switch (inputInt()) {

case 1:

sort\_getParametr(true);

break;

case 2:

sort\_getParametr(false);

break;

case 0:

dataProcessing();

default: {

cout << endl << "Вы ввели неправильный номер, попробуйте ещё раз";

Sleep(5000);

}

}

}

}

void viewData() {

system("cls");

cout << "Содержимое файла о сотрудниках:" << endl;

outputData();

cout << endl << "Нажмите Esc, чтобы выйти в предыдущее меню" << endl;

while (true)

if (\_getch() == 27)

if (roleOfUser)

modifyData();

else

return;

}

void addData() {

system("cls");

cout << "Добавление сотрудника" << endl;

addEmployee();

Employees[countOfEmployees - 1].id = getFreeID();

cout << "Введите имя: ";

Employees[countOfEmployees - 1].name = input();

cout << endl << "Введите фамилию: ";

Employees[countOfEmployees - 1].surname = input();

cout << endl << "Введите отчество: ";

Employees[countOfEmployees - 1].middleName = input();

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

if (tolower(Employees[i].name) == tolower(Employees[countOfEmployees - 1].name) && tolower(Employees[i].surname) == tolower(Employees[countOfEmployees - 1].surname) && tolower(Employees[i].middleName) == tolower(Employees[countOfEmployees - 1].middleName))

Employees[countOfEmployees - 1].tabelNumber = Employees[i].tabelNumber;

Employees[countOfEmployees - 1].tabelNumber == -1 ? getFreeTabelNumber() : Employees[countOfEmployees - 1].tabelNumber;

Employees[countOfEmployees - 1].year = inputYear\_Data(getLocalYear());

Employees[countOfEmployees - 1].month = inputMonth\_Data();

Employees[countOfEmployees - 1].countOfHours = inputCountOfHours\_Data();

Employees[countOfEmployees - 1].rate = inputRate\_Data();

editsInStructures = true;

cout << endl << "Сотрудник успешно добавлен в систему";

updateMaxLengthOfDataDatabase();

Sleep(5000);

modifyData();

}

void deleteData() {

system("cls");

cout << "Удаление сотрудника" << endl;

outputData();

int id = getID();

countOfEmployees--;

employee\* temp = new employee[countOfEmployees];

int j = 0;

for (int i = 0; i <= countOfEmployees; i++) {

if (Employees[i].id == id)

continue;

temp[j++] = Employees[i];

}

swap(temp, Employees);

editsInStructures = true;

cout << endl << "Удаление завершено";

updateMaxLengthOfDataDatabase();

Sleep(5000);

modifyData();

}

void editData() {

system("cls");

cout << "Изменение данных сотрудника" << endl;

outputData();

int id = getID(), i;

for (i = 0; i < countOfEmployees; i++)

if (id == Employees[i].id)

break;

while (true) {

int menu;

cout << endl << "Выбранный сотрудник: " << endl;

printf("%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s", to\_string(countOfEmployees).length(), "#", 3, " | ", maxLengthInDataDatabase[0], "Имя", 3, " | ", maxLengthInDataDatabase[1], "Фамилия", 3, " | ", maxLengthInDataDatabase[2], "Отчество", 3, " | ", maxLengthInDataDatabase[3], "Табельный номер", 3, " | ", maxLengthInDataDatabase[4], "Год", 3, " | ", maxLengthInDataDatabase[5], "Месяц", 3, " | ", maxLengthInDataDatabase[6], "Количество проработанных часов");

cout << " | Почасовой тариф" << endl;

outputOneData(i);

cout << "Выберите данные для изменения: " << endl << "1 - Имя" << endl << "2 - Фамилия" << endl << "3 - Отчество" << endl << "4 - Табельный номер" << endl << "5 - Год" << endl << "6 - Месяц" << endl << "7 - Количество проработанных часов" << endl << "8 - Почасовой тариф" << endl << "9 - Выбрать другого сотрудника" << endl << "0 - Выйти из данного режима" << endl << "Ваш вариант: ";

switch (menu = inputInt()) {

case 1: {

cout << endl << "Введите имя: ";

Employees[i].name = input();

}

break;

case 2: {

cout << endl << "Введите фамилия: ";

Employees[i].surname = input();

}

break;

case 3: {

cout << endl << "Введите отчество: ";

Employees[i].middleName = input();

}

break;

case 4: {

while (true) {

cout << endl << "Введите табельный номер: ";

Employees[i].tabelNumber = inputInt();

bool find = false;

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

if (Employees[i].tabelNumber == Employees[j].tabelNumber) {

find = true;

break;

}

if (!find)

break;

else

cout << endl << "Вы ввели неправильный табельный номер, попробуйте ещё раз";

}

}

break;

case 5:

Employees[i].year = inputYear\_Data(getLocalYear());

break;

case 6:

Employees[i].month = inputMonth\_Data();

break;

case 7:

Employees[i].countOfHours = inputCountOfHours\_Data();

break;

case 8:

Employees[i].rate = inputRate\_Data();

break;

case 9:

editData();

case 0:

modifyData();

default:

cout << endl << "Вы ввели неправильный номер, попробуйте ещё раз";

}

if (menu > 0 && menu < 9) {

cout << endl << "Запись о сотруднике успешно изменена";

editsInStructures = true;

updateMaxLengthOfDataDatabase();

Sleep(5000);

}

}

}

void outputData() {

printf("%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*s", to\_string(countOfEmployees).length(), "#", 3, " | ", maxLengthInDataDatabase[0], "Имя", 3, " | ", maxLengthInDataDatabase[1], "Фамилия", 3, " | ", maxLengthInDataDatabase[2], "Отчество", 3, " | ", maxLengthInDataDatabase[3], "Табельный номер", 3, " | ", maxLengthInDataDatabase[4], "Год", 3, " | ", maxLengthInDataDatabase[5], "Месяц", 3, " | ", maxLengthInDataDatabase[6], "Количество проработанных часов");

cout << " | Почасовой тариф" << endl;

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

outputOneData(i);

}

void outputOneData(int index) {

printf("%\*i%\*s%\*s%\*s%\*s%\*s%\*s%\*s%\*d%\*s%\*i%\*s%\*i%\*s%\*f", to\_string(countOfEmployees).length(), Employees[index].id, 3, " | ", maxLengthInDataDatabase[0], Employees[index].name.c\_str(), 3, " | ", maxLengthInDataDatabase[1], Employees[index].surname.c\_str(), 3, " | ", maxLengthInDataDatabase[2], Employees[index].middleName.c\_str(), 3, " | ", maxLengthInDataDatabase[3], Employees[index].tabelNumber, 3, " | ", maxLengthInDataDatabase[4], Employees[index].year, 3, " | ", maxLengthInDataDatabase[5], Employees[index].month, 3, " | ", maxLengthInDataDatabase[6], Employees[index].countOfHours);

cout << " | " << Employees[index].rate << endl;

}

int getLocalYear() {

time\_t now = time(0);

string localTime = string(ctime(&now));

string year;

for (int i = localTime.length() - 5; i < localTime.length(); i++)

year.push\_back(localTime[i]);

return stoi(year);

}

string tolower(string line) {

string result;

for (int i = 0; i < line.length(); i++) {

if ((line[i] >= 'A' && line[i] <= 'Z') || (line[i] >= 128 && line[i] <= 143)) {

result.push\_back(line[i] + 32);

continue;

}

if ((line[i] >= 'a' && line[i] <= 'z') || (line[i] >= 160 && line[i] <= 175) || (line[i] >= 224 && line[i] <= 239) || line[i] == 241) {

result.push\_back(line[i]);

continue;

}

if (line[i] >= 144 && line[i] <= 159) {

result.push\_back(line[i] + 80);

continue;

}

if (line[i] == 240)

result.push\_back(line[i]);

}

return result;

}

int getFreeTabelNumber() {

int\* array = new int[countOfEmployees];

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

\*(array + i) = Employees[i].tabelNumber;

for (int i = 0; i < countOfEmployees - 1; i++)

for (int j = i; j < countOfEmployees; j++)

if (\*(array + i) > \*(array + j))

swap(\*(array + i), \*(array + j));

int j = 0;

for (int i = 1; i <= INT\_MAX; i++) {

if (\*(array + j) == i) {

j++;

continue;

}

delete[] array;

return i;

}

}

int getFreeID() {

int\* array = new int[countOfEmployees];

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

\*(array + i) = Employees[i].id;

for (int i = 0; i < countOfEmployees - 1; i++)

for (int j = i; j < countOfEmployees; j++)

if (\*(array + i) > \*(array + j))

swap(\*(array + i), \*(array + j));

int j = 0;

for (int i = 1; i <= INT\_MAX; i++) {

if (\*(array + j) == i) {

j++;

continue;

}

delete [] array;

return i;

}

}

void updateMaxLengthOfDataDatabase() {

maxLengthInDataDatabase[0] = 3;

maxLengthInDataDatabase[1] = 7;

maxLengthInDataDatabase[2] = 8;

maxLengthInDataDatabase[3] = 15;

maxLengthInDataDatabase[4] = 3;

maxLengthInDataDatabase[5] = 5;

maxLengthInDataDatabase[6] = 30;

for (int i = 0; i < 7; i++)

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

switch (i) {

case 0:

maxLengthInDataDatabase[0] = maxLengthInDataDatabase[0] < Employees[j].name.length() ? Employees[j].name.length() : maxLengthInDataDatabase[0];

break;

case 1:

maxLengthInDataDatabase[1] = maxLengthInDataDatabase[1] < Employees[j].surname.length() ? Employees[j].surname.length() : maxLengthInDataDatabase[1];

break;

case 2:

maxLengthInDataDatabase[2] = maxLengthInDataDatabase[2] < Employees[j].middleName.length() ? Employees[j].middleName.length() : maxLengthInDataDatabase[2];

break;

case 3:

maxLengthInDataDatabase[3] = maxLengthInDataDatabase[3] < to\_string(Employees[j].tabelNumber).length() ? to\_string(Employees[j].tabelNumber).length() : maxLengthInDataDatabase[3];

break;

case 4:

maxLengthInDataDatabase[4] = maxLengthInDataDatabase[4] < to\_string(Employees[j].year).length() ? to\_string(Employees[j].year).length() : maxLengthInDataDatabase[4];

break;

case 5:

maxLengthInDataDatabase[5] = maxLengthInDataDatabase[5] < to\_string(Employees[j].month).length() ? to\_string(Employees[j].month).length() : maxLengthInDataDatabase[5];

break;

case 6:

maxLengthInDataDatabase[6] = maxLengthInDataDatabase[6] < to\_string(Employees[j].countOfHours).length() ? to\_string(Employees[j].countOfHours).length() : maxLengthInDataDatabase[6];

}

}

double getSalary(int i, double countOfHours) {

double salary = 0;

if (countOfHours > 144) {

salary += 2 \* ((countOfHours - 144) \* Employees[i].rate);

salary += 144 \* Employees[i].rate;

}

else

salary += countOfHours \* Employees[i].rate;

salary -= salary \* 0.12;

return salary;

}

void sort\_getParametr(bool typeOfSort /\*1 - bubble, 0 - quick\*/) {

int criteria;

while (true) {

cout << (typeOfSort ? "Пузырьковая" : "Быстрая") << " сортировка | Выберите параметр" << endl;

showCriteria();

cout << "Ваш выбор: ";

switch (criteria = inputInt()) {

case 1:

case 2:

case 3:

case 4:

case 5:

case 6:

break;

default:

cout << endl << "Вы ввели неправильный номер, попробуйте ещё раз";

}

if (criteria > 0 && criteria < 7)

break;

}

typeOfSort ? bubbleSort(criteria) : quickSort(0, countOfEmployees - 1, criteria);

cout << endl << "Сортировка выполнена" << endl;

cout << "Хотите выйти в предыдущее меню или выполнить сортировку ещё раз? (0 - выход в предыдущее меню, 1 - меню выбора параметра сортировки): ";

inputBool() ? sort\_getParametr(typeOfSort) : sort();

}

void showCriteria() {

cout << "1 - По номеру записи" << endl;

cout << "2 - По табельному номеру" << endl;

cout << "3 - По году" << endl;

cout << "4 - По месяцу" << endl;

cout << "5 - По количеству проработанных часов" << endl;

cout << "6 - По почасовому тарифу" << endl;

}

void bubbleSort(int criteria) {

for (int i = 0; i < countOfEmployees - 1; i++) {

for (int j = i; j < countOfEmployees; j++) {

bool needToSwap = false;

switch (criteria) {

case 1:

needToSwap = Employees[i].id > Employees[j].id;

break;

case 2:

needToSwap = Employees[i].tabelNumber > Employees[j].tabelNumber;

break;

case 3:

needToSwap = Employees[i].year > Employees[j].year;

break;

case 4:

needToSwap = Employees[i].month > Employees[j].month;

break;

case 5:

needToSwap = Employees[i].countOfHours > Employees[j].countOfHours;

break;

case 6:

needToSwap = Employees[i].rate > Employees[j].rate;

}

if (needToSwap)

swap(Employees[i], Employees[j]);

}

}

}

void quickSort(int leftIndex, int rightIndex, int criteria) {

if (rightIndex > leftIndex) {

int middleIndex = getMiddleIndex(leftIndex, rightIndex, criteria);

quickSort(leftIndex, middleIndex - 1, criteria);

quickSort(middleIndex + 1, rightIndex, criteria);

}

}

int getMiddleIndex(int leftIndex, int rightIndex, int criteria) {

int middleIndex = rightIndex, i = leftIndex - 1;

for (int j = leftIndex; j <= rightIndex - 1; j++) {

bool needToSwap = false;

switch (criteria) {

case 1:

needToSwap = Employees[j].id <= Employees[middleIndex].id;

break;

case 2:

needToSwap = Employees[j].tabelNumber <= Employees[middleIndex].tabelNumber;

break;

case 3:

needToSwap = Employees[j].year <= Employees[middleIndex].year;

break;

case 4:

needToSwap = Employees[j].month <= Employees[middleIndex].month;

break;

case 5:

needToSwap = Employees[j].countOfHours <= Employees[middleIndex].countOfHours;

break;

case 6:

needToSwap = Employees[j].rate <= Employees[middleIndex].rate;

}

if (needToSwap)

swap(Employees[j], Employees[++i]);

}

swap(Employees[++i], Employees[rightIndex]);

return i;

}

string getTempParametr(employee employee, int criteria) {

switch (criteria) {

case 1:

return to\_string(employee.id);

case 2:

return to\_string(employee.tabelNumber);

case 3:

return to\_string(employee.year);

case 4:

return to\_string(employee.month);

case 5:

return round(employee.countOfHours) == employee.countOfHours ? to\_string(int(round(employee.countOfHours))) : to\_string(employee.countOfHours);

case 6:

return round(employee.rate) == employee.rate ? to\_string(int(round(employee.rate))) : to\_string(employee.rate);

}

}

int getID() {

int id;

while (true) {

cout << "Введите номер записи: ";

id = inputInt();

if (id >= 0 && id <= getMaxID())

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

if (id == Employees[i].id)

return id;

cout << endl << "Вы ввели неверный номер, попробуйте ещё раз" << endl;

}

}

int getMaxID() {

int max = INT\_MIN;

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

max = max < Employees[i].id ? Employees[i].id : max;

return max;

}

# **ПРИЛОЖЕНИЕ Б**

#ifndef INPUT\_H

#define INPUT\_H

#undef max

#include <iostream>

#include <string>

#include <conio.h>

#include <limits>

using namespace std;

string input() { // функция ввода строк

string line;

while (getline(cin, line))

if (!line.empty())

return line;

}

// функция, в которой пользователь вводит логин

string inputLogin() {

string validChars = "-.0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ\_abcdefghijklmnopqrstuvwxyz"; // допустимые символы в логине

while (true) {

string login; // строка для вводимого логина

bool allIsGood = true; // логическая переменная, в которой отслеживается всё ли хорошо с логином

while (getline(cin, login))

if (!login.empty()) {

for (int i = 0; i < login.length(); i++)

if (validChars.find(login[i]) == string::npos) {

allIsGood = false;

break;

}

if (allIsGood)

return login;

else

break;

}

login.clear();

cout << endl << "Логин содержит недопустимые символы, попробуйте ещё раз" << endl << "Введите логин: ";

}

}

// функция, аналогичная inputLogin, но здесь пользователь вводит пароль

string inputPassword() {

string validChars = "!\"#$%&'()\*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~";

while (true) {

string password;

bool allIsGood = true;

while (getline(cin, password))

if (!password.empty()) {

for (int i = 0; i < password.length(); i++)

if (validChars.find(password[i]) == string::npos) {

allIsGood = false;

break;

}

if (allIsGood)

return password;

else

break;

}

password.clear();

cout << endl << "Пароль содержит недопустимые символы, попробуйте ещё раз" << endl << "Введите пароль: ";

}

}

string inputPassword\_hide() { // функция, аналогичная inputPassword, но на экране пользователь видит звёздочки, вместо символов пароля

string password; // строка, хранящая пароль, вводимый пользователем

int count = 0; // целочисленная переменная, контролирующая пределы вводимого пароля

string validChars = "!\"#$%&'()\*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~"; // допустимые символы

while (true) {

char temp = \_getch();

if (short(temp) == 13 || count == 255) {

if (password.empty()) {

cout << endl << "Вы не ввели пароль, попробуйте ещё раз" << endl << "Введите пароль: ";

password.clear();

count = 0;

continue;

}

else {

bool allIsGood = true; // логическая переменная, в которой отслеживается всё ли хорошо с логином

for (int i = 0; i < password.length(); i++)

if (validChars.find(password[i]) == string::npos) {

allIsGood = false;

break;

}

if (allIsGood)

return password;

else {

cout << endl << "Пароль содержит недопустимые символы, попробуйте ещё раз" << endl << "Введите пароль: ";

password.clear();

count = 0;

continue;

}

}

}

password.push\_back(temp);

count++;

cout << "\*";

}

}

bool inputBool() { // функция ввода значений типа bool

while (true) {

bool temp; // создаём временную переменную для ввода

cin >> temp; // вводим значение

if (cin.good()) // если пользователь ввел вреное значение

return temp; // возвращаем значение

else { // иначе

cin.clear(); // сбрасываем флаги ошибок

cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // очистка потока

cout << endl << "Некорректный ввод, попробуйте ещё раз" << endl << "Введите значение: "; // вывод сообщения об ошибке

}

}

}

int inputInt() { // функция ввода значений типа int

while (true) {

int temp; // создаём временную переменную для ввода

cin >> temp; // вводим значение

if (cin.good()) // если пользователь ввел вреное значение

return temp; // возвращаем значение

else { // иначе

cin.clear(); // сбрасываем флаги ошибок

cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // очистка потока

cout << endl << "Некорректный ввод, попробуйте ещё раз" << endl << "Введите значение: "; // вывод сообщения об ошибке

}

}

}

double inputDouble() { // функция ввода значений типа double

while (true) {

double temp; // создаём временную переменную для ввода

cin >> temp; // вводим значение

if (cin.good()) // если пользователь ввел вреное значение

return temp; // возвращаем значение

else { // иначе

cin.clear(); // сбрасываем флаги ошибок

cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // очистка потока

cout << endl << "Некорректный ввод, попробуйте ещё раз" << endl << "Введите значение: "; // вывод сообщения об ошибке

}

}

}

int inputYear\_Data(int localYear) { // функция ввода года

while (true) {

cout << endl << "Введите год: ";

int year = inputInt(); // переменная, в которую пользователь вводит год

if (year > -1 && year <= localYear) // проверка года на действительность

return year;

else

cout << endl << "Вы ввели неправильный год, попробуйте ещё раз";

}

}

int inputMonth\_Data() { // функция ввода месяца

while (true) {

cout << endl << "Введите месяц: ";

int month = inputInt(); // переменная, в которую пользователь вводит месяц

if (month > 0 && month < 13) // проверка месяца на действительность

return month;

else

cout << endl << "Вы ввели неправильный месяц, попробуйте ещё раз";

}

}

double inputCountOfHours\_Data() { // функция ввода количества часов

while (true) {

cout << endl << "Введите количество проработанных часов: ";

double countOfHours = inputDouble(); // переменная, в которую пользователь вводит количество часов

if (countOfHours >= 0) // проверка колмчества на действительность

return countOfHours;

else

cout << endl << "Вы ввели неправильное количество проработанных часов, попробуйте ещё раз";

}

}

double inputRate\_Data() { // функция ввода тарифа

while (true) {

cout << endl << "Введите почасовой тариф: ";

double rate = inputDouble(); // переменная, в которую пользователь вводит тариф

if (rate >= 0) // проверка тарифа на действительность

return rate;

else

cout << endl << "Вы ввели неправильный почасовой тариф, попробуйте ещё раз";

}

}

#endif //INPUT\_H

# ПРИЛОЖЕНИЕ В

#ifndef CRYPTO\_H

#define CRYPTO\_H

#include <iostream>

using namespace std;

string normalLine = "!\"#$%&'()\*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~"; // строка нешифрованных символов

string cryptedLine = "mdl|cajoz~{wpyrusvktqhei`bnxf}gUHF[WPMR^GCAITY\\_]OEXSBJNKQ@VDLZ.\"\*?<-50&$;/:%'#2,7(>!8)14=3+69"; // строка шифрованных символов

string crypt(string line) { // функция, шифрующая строку

string result; // строка для шифрованной строки

for (int i = 0; i < line.length(); i++) // цикл по всей нешифрованной строке

result.push\_back(cryptedLine[normalLine.find(line[i])]); // добавляем в конец шифрованной строки символ, который у нас шифруется

return result; // возвращаем зашифрованную строку

}

string decrypt(string line) { // аналогичная функция, дешифрующая строку

string result;

for (int i = 0; i < line.length(); i++)

result += normalLine[cryptedLine.find(line[i])];

return result;

}

#endif //CRYPTO\_H