Содержание

|  |  |
| --- | --- |
|  |  |
| Введение | 3 |
| 1 Описание предметной области | 4 |
| 1.1 Постановка задачи | 4 |
| 1.2 Объектно-ориентированный анализ предметной области | 5 |
| 2 Проектирование программного средства | 6 |
| 2.1 Проектирование структур данных | 6 |
| 2.2 Проектирование взаимодействия с пользователем | 8 |
| 2.3 Проектирование интерфейса | 9 |
| 3 Реализация программного средства | 11 |
| 3.1 Реализация классов и структур данных | 11 |
| 3.2 Реализация требований | 13 |
| 3.3 Реализация концепций ООП | 13 |
| 3.4 Описание структуры проекта | 13 |
| 3.5 Реализация интерфейса взаимодействия с пользователем | 14 |
| 4 Тестирование программного модуля | 16 |
| Заключение | 23 |
| Список использованных источников | 24 |
| Приложение А. Код программы | 25 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Введение

Целью курсовой работы является создание программного обеспечения для ведения учета заработной платы сотрудников предприятия.

Задачи курсовой работы: разработать программное обеспечение, протестировать программное обеспечение, описать работу программного обеспечения.

Ожидаемый результат курсового проекта: должно быть полное рабочее программное обеспечение. Оно должно выводить всю информацию о заработной плате сотрудников. Добавлять, удалять, сортировать записи. Должно быть подробное и понятное меню для пользователя. Также должна содержаться кнопка выхода из программного обеспечения. Программа должна работать корректно, исключая ошибки ввода от пользователя.

В пояснительной записке описан процесс создания программного продукта, описаны принципы его работы и результаты тестирования.

1 Описание предметной области

* 1. Постановка задачи

Целью курсовой работы является создание программного обеспечения на языке С++ для ведения учета заработной платы сотрудников предприятия.

Данные программы должны храниться в файлах. Разрабатываемое меню должно быть понятным и защищенным от случайных ошибок, вылетов и ввода некорректных данных. Ввод исходных данных должен осуществляться с клавиатуры.

Разрабатываемая программа должна обеспечить возможность выполнения перечисленных ниже функций:

* вывод записей о заработной плате в консоль;
* добавление записей о заработной плате;
* удаление записей;
* сортировка записей по дате создания, имени и фамилии сотрудника;
* чтение из файла;
* запись в файл.

Требования к организации ввода входных данных:

* Входные данные должны соответствовать меню программного обеспечения (одна из цифр пункта меню)

Требования к организации вывода входных данных:

Информация о записях в виде сгруппированной текстовой информации в консоли.

Тип пользовательского интерфейса-консольное приложение.

Функциональные требования к разрабатываемому ПО представлены на рисунке 1.1.

При выполнении программы необходимо предусмотреть обработку следующих ошибок:

* несоответствие номера пункта меню
* неправильный формат исходных данных

Функциональные требования представлены на рисунке 1.1 в виде диаграммы прецедентов.

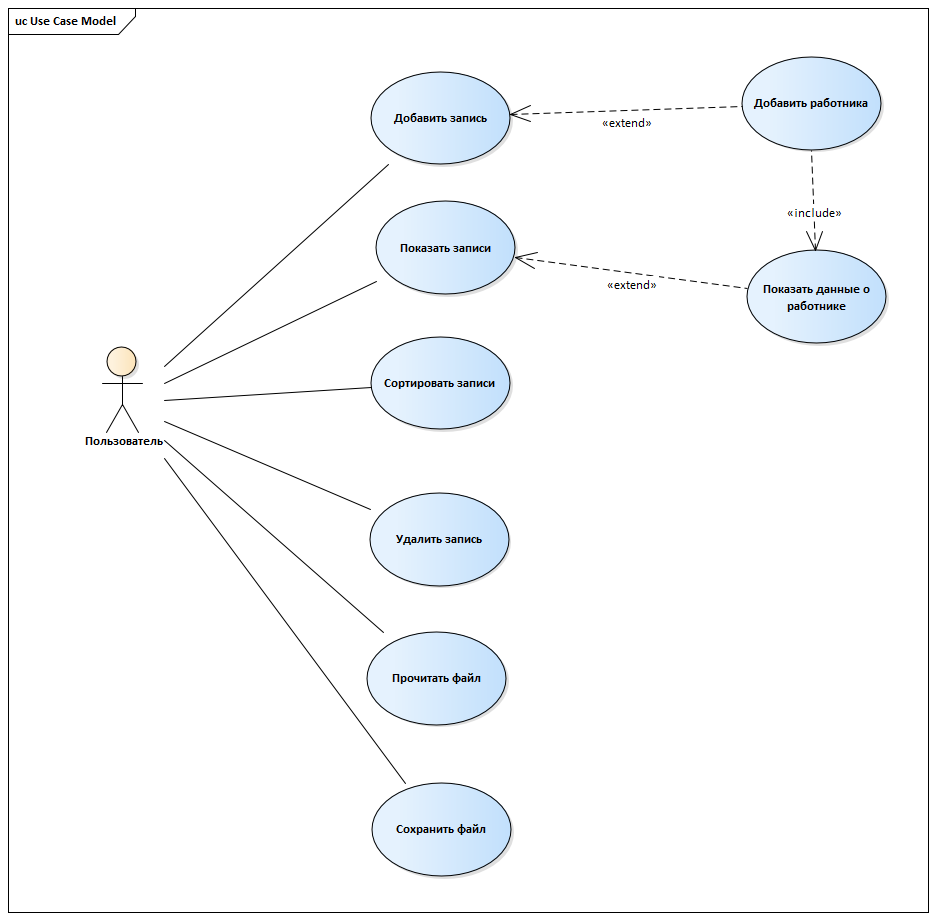


Рисунок 1.1 – Диаграмма вариантов использования

1.2 Объектно-ориентированный анализ предметной области

Запись характеризуется датой, сотрудником.

Дата характеризуется годом, месяцем, числом создания.

Сотрудник предприятия характеризуется именем, фамилией, отчеством, адресом, контактами, должностью, отработанными часами, заработной платой.

2.Проектирование программного средства

* 1. Проектирование структур данных

В соответствии с пунктом 1.2 для реализации предметной области выделены классы:

* запись;
* сотрудник;
* дата;

Для реализации хранения данных предусмотрен класс Storage.

Меню организованно с использованием класса Menu.

В разрабатываемом ПО имеется один вектор: accounts.

Диаграмма классов представлена на рисунке 2.1.

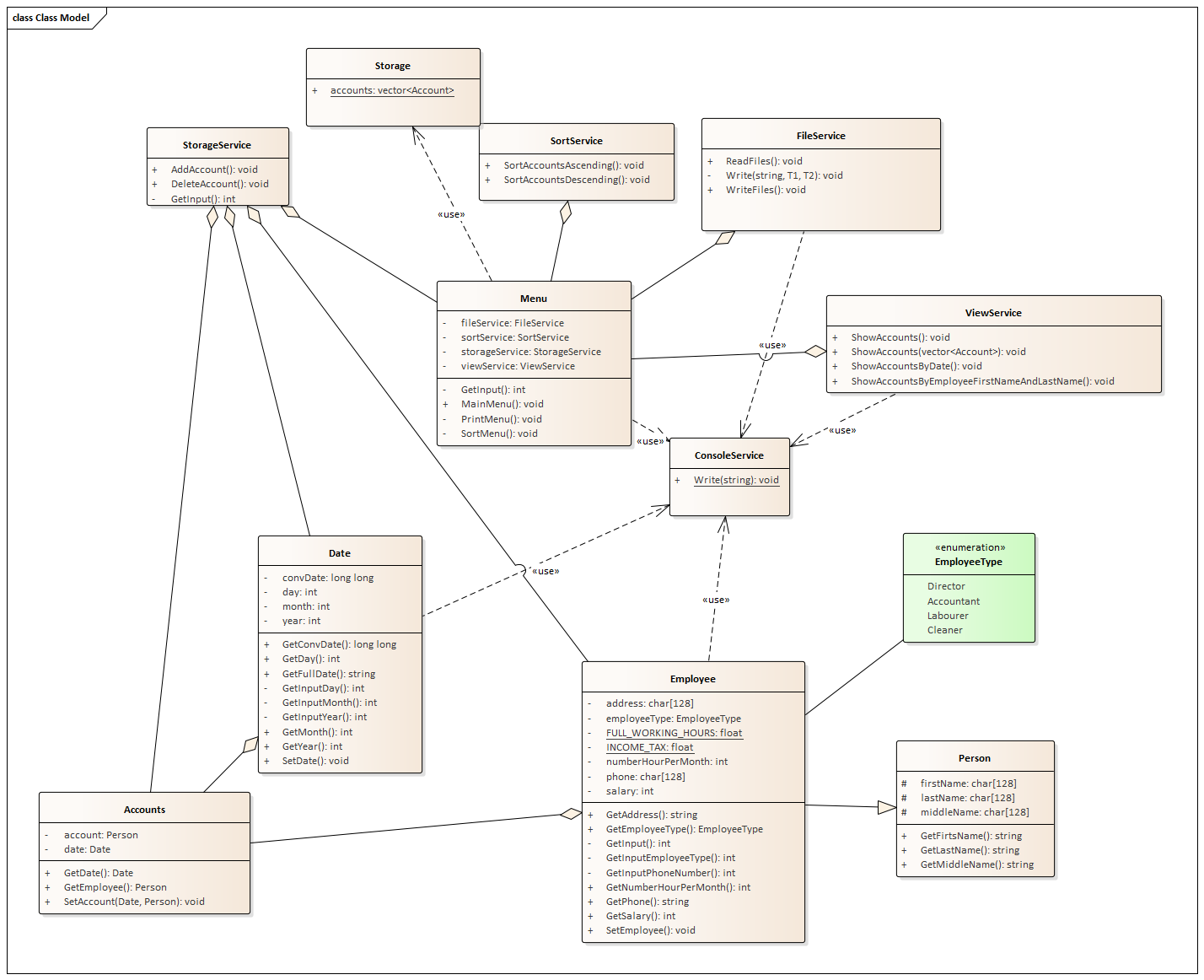


Рисунок 2.1 – Диаграмма классов

Структура классов хранящих данные предметной области представлена в таблицах 2.1-2.4.

Таблица 2.1 – Описание полей класса Запись

|  |  |  |
| --- | --- | --- |
| Имя поля | Тип данных | Ограничение |
| Дата | Date | - |
| Сотрудник | Employee | - |

Таблица 2.2 – Описание полей класса Сотрудник

|  |  |  |
| --- | --- | --- |
| Имя поля | Тип данных | Ограничение |
| Имя | char | Длина от 1 до 128 символов |
| Фамилия | char | Длина от 1 до 128 символов |
| Отчество | char | Положительное |
| Должность | EmployeeType | От 0 до 3 |
| Заработная плата | int | От -2 147 483 648 до 2 147 483 647 |
| Телефон | char | Длина от 1 до 128 символов |
| Адрес | char | Длина от 1 до 128 символов |
| Налог | float | От -3.4E-38 до 3.4E+38 |
| Количество отработанных часов | int | От -2 147 483 648 до 2 147 483 647 |

Таблица 2.3 – Описание полей класса Дата

|  |  |  |
| --- | --- | --- |
| Имя поля | Тип данных | Ограничение |
| Год | int | От -2 147 483 648 до 2 147 483 647 |
| Месяц | int | От -2 147 483 648 до 2 147 483 647 |
| День | int | Положительное |
| Полная дата | long long | От −9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 |

* 1. Проектирование взаимодействия с пользователем

Реализация прецедент Добавление записи представлена в виде диаграммы взаимодействия, рисунок 2.2.

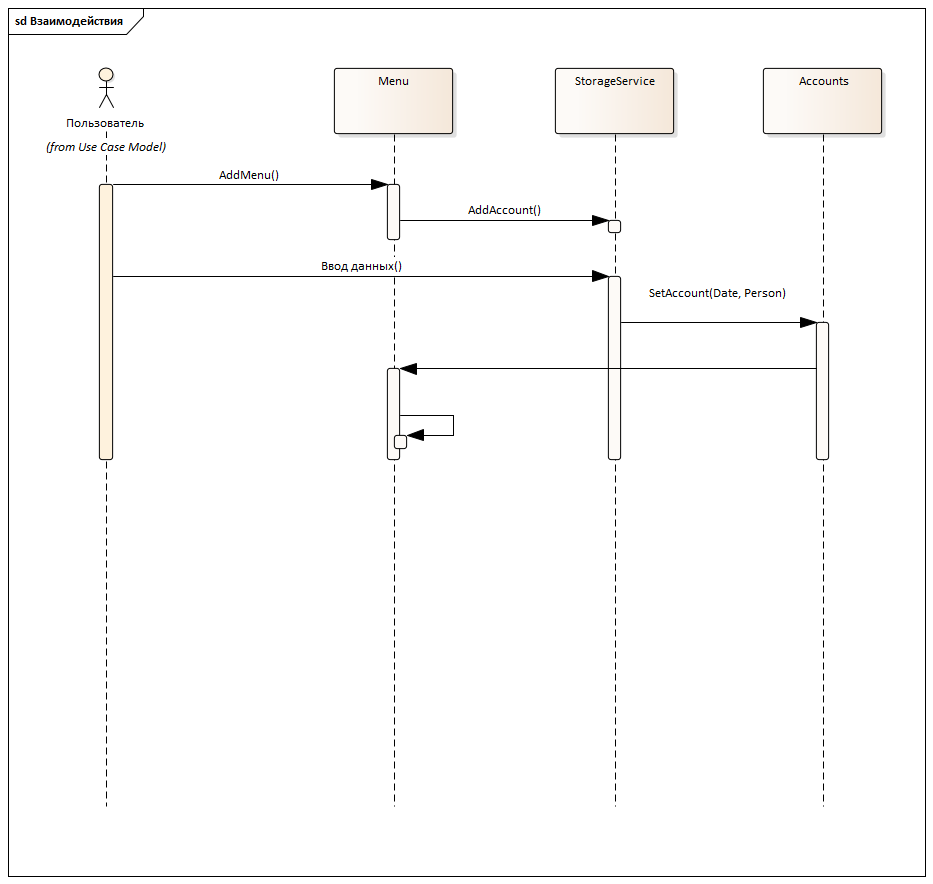


Рисунок 2.2 – прецедент Добавление записи

Реализация прецедент Чтение файла представлена в виде диаграммы взаимодействия, рисунок 2.3.

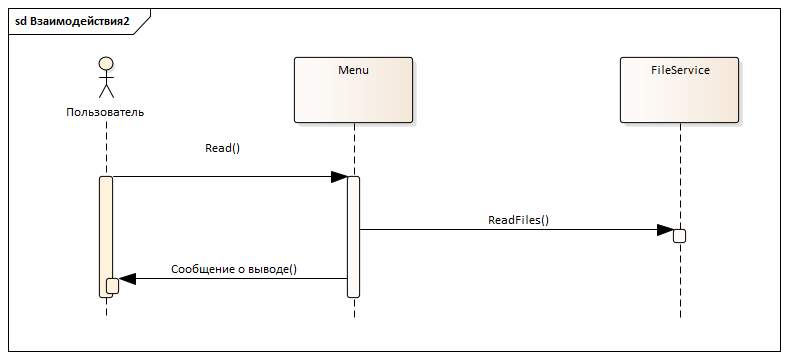


Рисунок 2.3 – прецедент Чтение файла

Реализация прецедент Сортировка записей по дате представлена в виде диаграммы взаимодействия, рисунок 2.4.

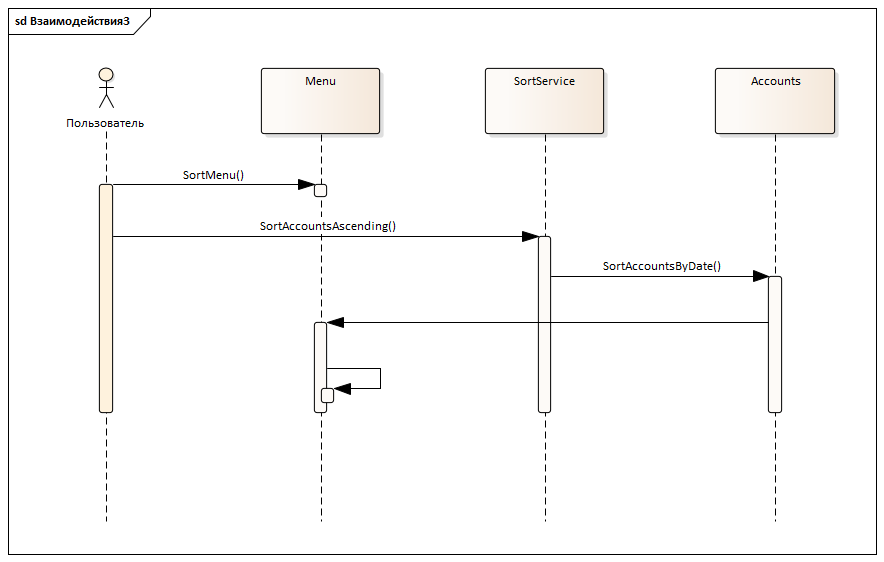


Рисунок 2.4 – прецедент Сортировка записей по дате

* 1. Проектирование интерфейса

Схема меню для работы с пользователем продемонстрирована на рисунке 2.3

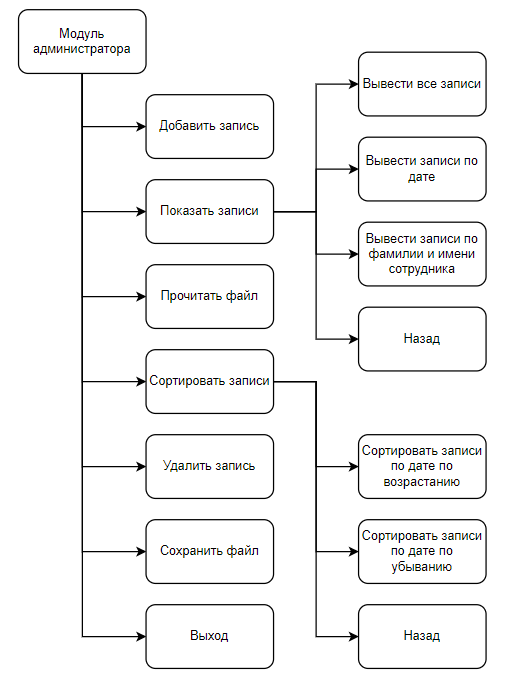


Рисунок 2.3 –Схема меню

1. Реализация программного средства
   1. Реализация классов и структур данных

Реализация класса Employee

private:

char address[128];

char phone[128];

int numberHourPerMonth;

int salary;

EmployeeType employeeType;

static constexpr float INCOME\_TAX = 0.12f;

void GetInput();

string GetInputPhoneNumber();

int GetInputEmployeeType();

public:

EmployeeType GetEmployeeType();

string GetAddress();

string GetPhone();

int GetNumberHourPerMonth();

int GetSalary();

void SetEmployee();

Методы GetFirstName(), GetLastName(), GetMiddleName() класс Employee наследует у класса Person.

Реализация класса Date

private:

int year = 0;

int month = 0;

int day = 0;

long long convDate = 0;

int GetInputYear();

int GetInputMonth();

int GetInputDay();

public:

int GetYear();

int GetMonth();

int GetDay();

void SetDate();

string GetFullDate();

long long GetConvDate();

Реализация класса Account

private:

Date date;

Employee employee;

public:

Date GetDate();

Employee GetEmployee();

void SetAccount(Date date, Employee employee) ;

Реализация класса Person

protected:

char firstName[128];

char lastName[128];

char middleName[128];

public:

string GetFirtsName();

string GetLastName();

string GetMiddleName();

Подробный код программы продемонстрирован в приложении А

3.2 Реализация требований

Реализация требований согласно пункту 1.1 представлена в таблице 3.1.

Таблица 3.1 – Реализация требований

|  |  |  |
| --- | --- | --- |
| Функциональное требование | Реализация | Примечание |
| Должна производиться сортировка записей по дате в порядке возрастания | Метод SortAccountsAscending() | Метод не принимает параметров |
| Должна производиться сортировка записей по дате в порядке убывания | Метод SortAccountsDescending() | Метод не принимает параметров |
| Должно производится удаление записей | Метод DeleteAccount() | Метод не принимает параметров |
| Должно производится добавление записей | Метод AddAccount() | Метод не принимает параметров |
| Должен производиться поиск записей по имени и фамилии сотрудника | Метод ShowAccountsByEmployee-FirstNameAndLastName () | Метод не принимает параметров |
| Должен производиться поиск записей по дате | Метод ShowAccountsByDate() | Метод не принимает параметров |
| Должен производиться вывод всех записей на экран | Метод ShowAccounts() | Метод не принимает параметров |

3.3 Реализация концепций ООП

Для реализации инкапсуляции используются классы. Для обеспечения сокрытия все поля имеют спецификатор доступа private. В public занесены все методы, которые необходимы для использования программы. Геттеры и сеттеры необходимы для чтения и записи переменных в класс и из класса.Перегрузка методов в программировании — один из способов реализации полиморфизма, а именно Ad-Hoc. Наследование позволяет строить иерархии, в которых классы-потомки получают свойства классов-предков и могут дополнять их или изменять.

3.4 Описание структуры проекта

Описание структуры проекта представлено на рисунке 3.4-1

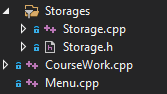
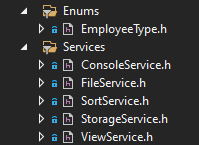
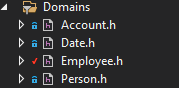


Рисунок 3.4-1 – Структура проекта

3.5 Реализация интерфейса взаимодействия с пользователем

При запуске программы пользователь увидит главное меню программы. (рисунок 3.1), в котором семь пунктов:

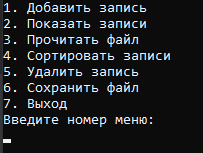


Рисунок 3.1 –Главное меню

При выборе пункта “Добавить запись” пользователь может заполнить новую запись. (рисунок 3.2)

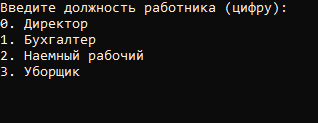


Рисунок 3.2 – Пункт “Добавить запись”

При выборе пункта “Показать записи” пользователь попадает в меню, в котором может выбрать что ему необходимо вывести. (рисунок 3.3)

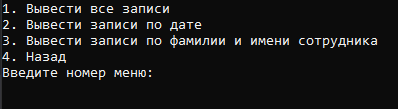


Рисунок 3.3 – Подменю пункта “Показать записи”

При выборе пункта “Прочитать файл” программа считывает данные из файла с названием “accounts.bin”.

При выборе пункта “Сортировать записи” пользователь попадает в подменю с выбором сортировки. (рисунок 3.4)

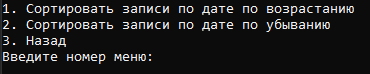


Рисунок 3.4 – Подменю пункта “Сортировать записи”

При выборе пункта “Удалить запись” пользователь может удалить запись. (Рисунок 3.5)

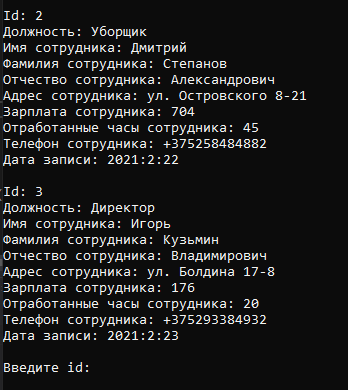


Рисунок 3.5 –Пункта “Удалить запись”

При выборе пункта “Сохранить файл” программа создаёт новый файл с названием “ accounts.bin”. (Рисунок 3.6)

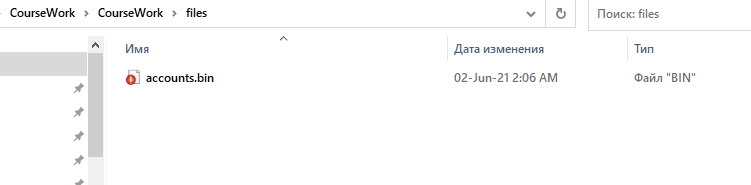


Рисунок 3.6 –Файл из папки проекта

При выборе пункта “Выход” программа закрывается.

4 Тестирование программного модуля

При запуске программы пользователь увидит главное меню программы. (рисунок 4.1), в котором семь пунктов:

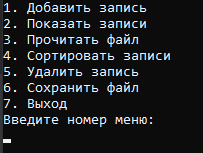


Рисунок 4.1 – Главное меню ПО

При выборе пункта “Добавить запись” пользователь может вводит данные для записи (рисунок 4.2.

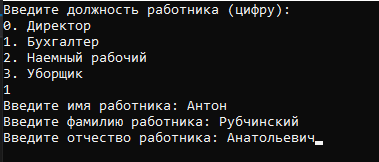


Рисунок 4.2 –Пункта “Добавить запись”

При выборе первого пункта пользователь попадает в окно внесения данных покупателя. (рисунок 4.3)

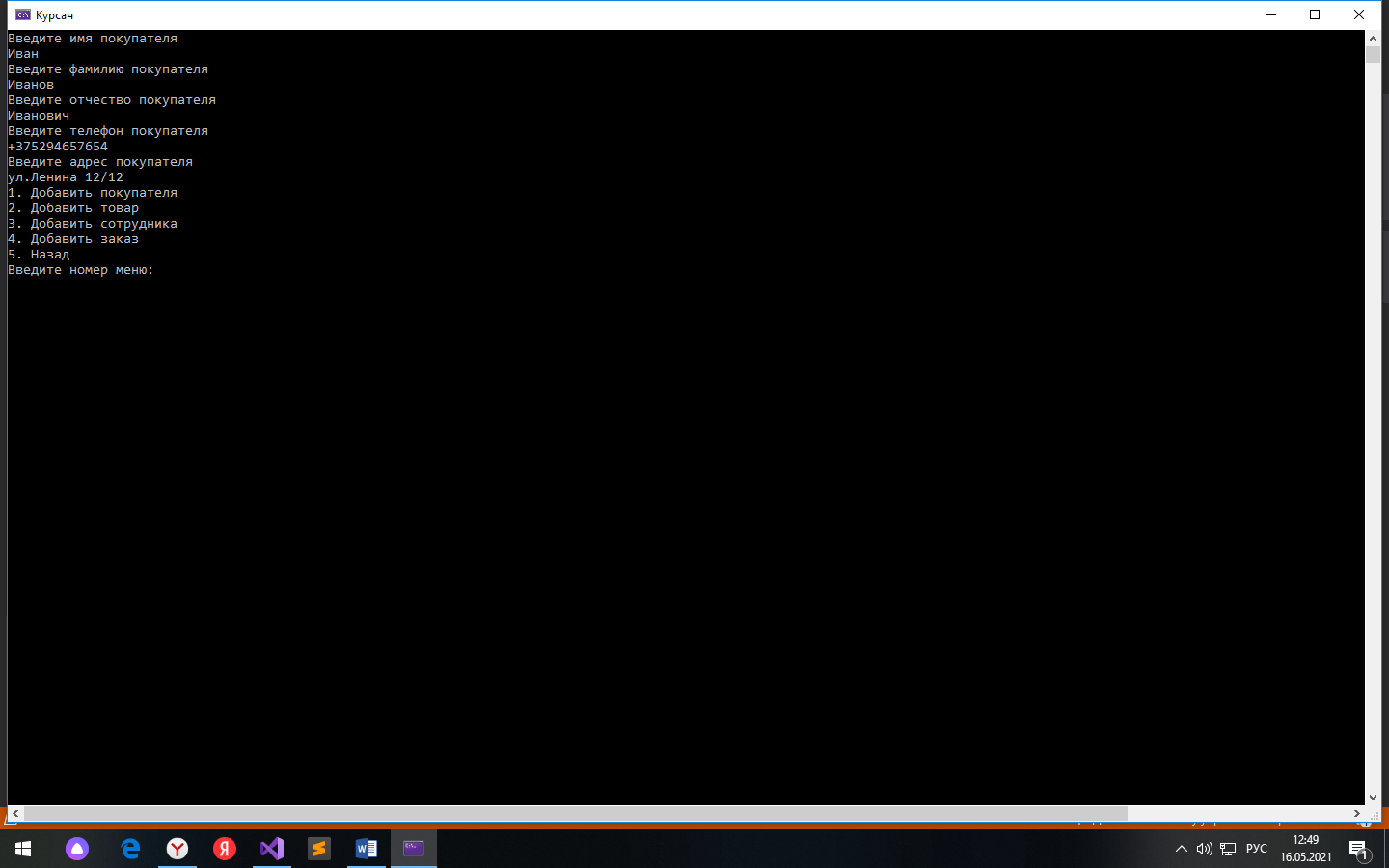


Рисунок 4.3 – Демонстрация внесения данных

При выборе пункта “Показать записи” пользователь попадает в меню, в котором может выбрать что ему необходимо вывести. (рисунок 4.4)

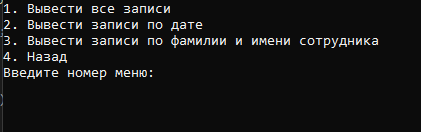


Рисунок 4.4 – Меню вывода данных

При выборе первого пункта пользователь получает данные о записях (рисунок 4.5)

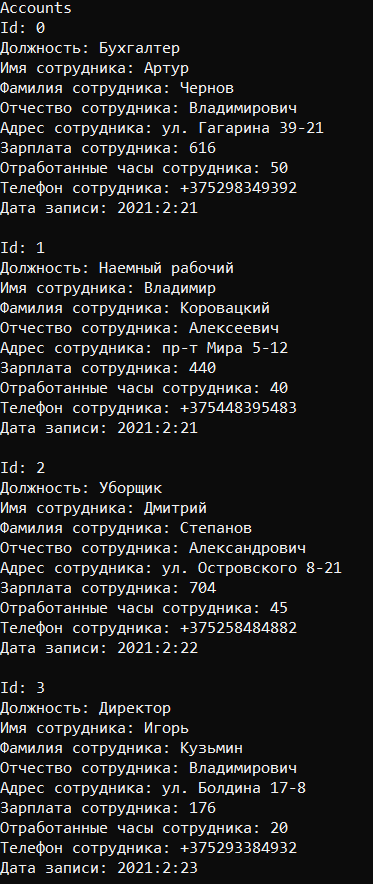


Рисунок 4.5 – Демонстрация вывода данных о сотрудниках

При выборе второго пункта пользователь получает данные о записях за определенную дату (рисунок 4.6)

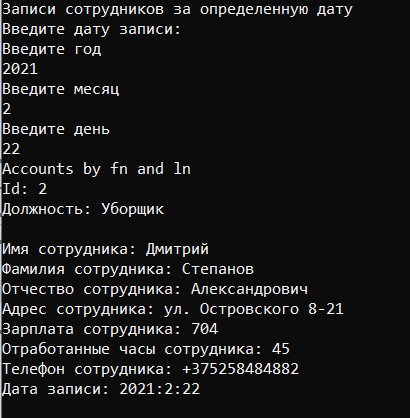


Рисунок 4.6 – Демонстрация вывода данных

При выборе третьего пункта пользователь получает данные о записях по фамилии и имени сотрудника (рисунок 4.7)

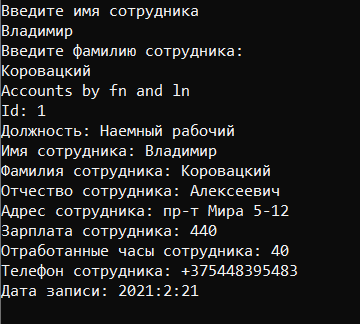


Рисунок 4.7 – Демонстрация вывода записей

При неправильном вводе будет выведено сообщение об ошибке. (рисунок 4.8)

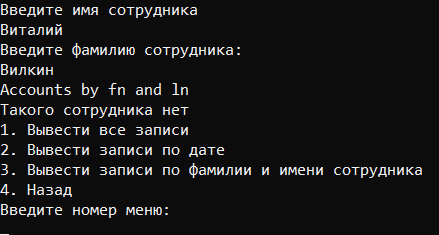


Рисунок 4.8 – Демонстрация обработки ошибок

При выборе пункта “Прочитать файл” программа считывает данные из файла с названием “accounts.bin”.

При выборе пункта “Сортировать записи” пользователь попадает в подменю с выбором сортировки. (рисунок 4.9)

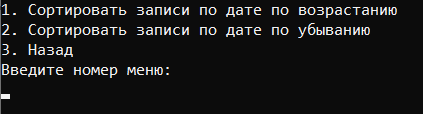


Рисунок 4.9 – Подменю пункта “Сортировать записи”

При выборе первого пункта, записи сортируются по возрастанию по дате. (рисунок 4.10)

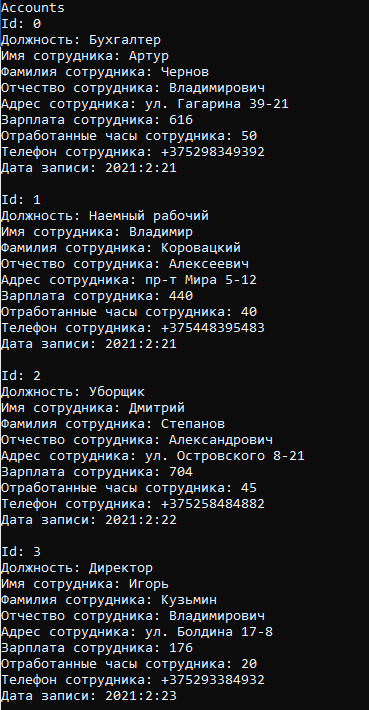


Рисунок 4.10 – Демонстрация сортировки по возрастанию

При выборе второго пункта, записи сортируются по убыванию. (рисунок 4.11)

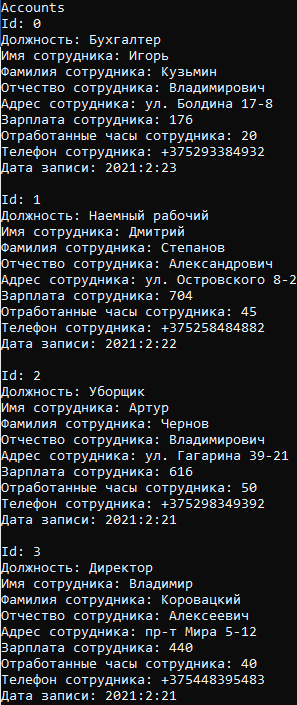


Рисунок 4.11 – Демонстрация сортировки по убыванию

При выборе пункта “Удалить запись” пользователь выбирает id записи (Рисунок 4.12), которой он хочет удалить

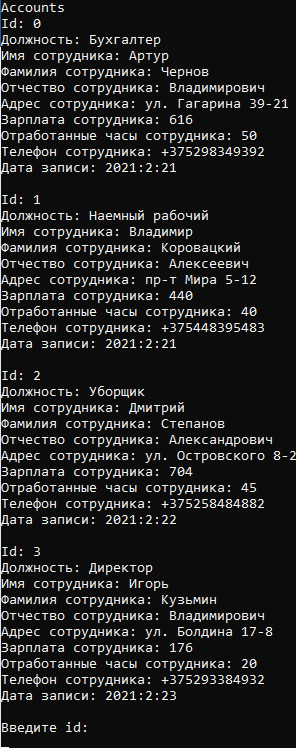


Рисунок 4.12 – Пункт меню “Удалить запись”

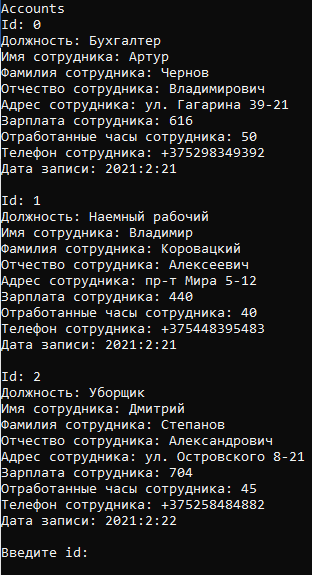


Рисунок 4.13 – Демонстрация удаления покупателя

При выборе пункта “Сохранить файл” программа создаёт новый файл с названием “accounts.bin” (Рисунок 4.21), где данные записываются в этот файл, которые можно прочесть или использовать при новом запуске программы. Если эти файлы уже созданы, то программа перезаписывает данные в файлы.

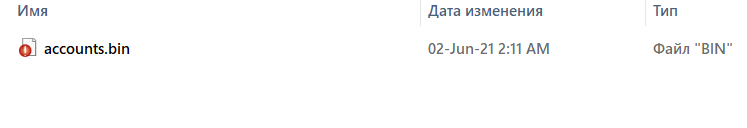


Рисунок 4.21 – Файл из папки проекта

При выборе пункта “Выход” программа закрывается.

# Заключение

В ходе проделанной работы была разработана программа для учета заработной платы сотрудников предприятия. Корректность работы программы подтверждено тестами.

В процессе выполнения работы были закреплены знания программирования на языке С++.Также получены практические навыки. Углублены знания от постановки задачи до ее реализации.

Данная программа предназначена для работы с данными сотрудников предприятия.

Достоинства данного программного обеспечения - это простой и понятный интерфейс для пользователя, множество проверок на корректность ввода, что исключает ошибки.

Список использованных источников

1. <https://metanit.com/>

2. <https://www.cyberforum.ru/>

4. <https://ravesli.com/>

5. https://www.microsoft.com/

6. Васильев, А.Н. Программирование на C++ в примерах и задачах / А.Н. Васильев. - М.: ЭКСМО, 2017. - 416 c.

7. Методические указания к курсовому проектированию для студентов специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)» дневной и заочной форм обучения

Приложение А

Код программы

Account.h

#pragma once

#include <iostream>

#include <string>

#include "Date.h"

#include "Employee.h"

using namespace std;

class Account

{

private:

Date date;

Employee employee;

public:

Date GetDate()

{

return date;

}

Employee GetEmployee()

{

return employee;

}

void SetAccount(Date date, Employee employee)

{

this->date = date;

this->employee = employee;

}

};

Date.h

#pragma once

#include <iostream>

#include <regex>

#include "ConsoleService.h"

using namespace std;

class Date

{

private:

int year = 0;

int month = 0;

int day = 0;

long long convDate = 0;

int GetInputYear()

{

string yy;

int year;

bool isCorrectInput = false;

while (!isCorrectInput)

{

cin >> yy;

regex regular("(\\d{4})");

try

{

if (!regex\_match(yy, regular))

{

throw 0;

}

year = stoi(yy);

if (year > 2022 || year < 2000)

{

throw 0;

}

}

catch (...)

{

ConsoleService::Write("Неправильный ввод\n");

isCorrectInput = false;

continue;

}

isCorrectInput = true;

}

return year;

}

int GetInputMonth()

{

string mm;

int month;

bool isCorrectInput = false;

while (!isCorrectInput)

{

cin >> mm;

regex regular("(\\d{2}||\\d{1})");

try

{

if (!regex\_match(mm, regular))

{

throw 0;

}

month = stoi(mm);

if (month > 12 || month < 1)

{

throw 0;

}

}

catch (...)

{

ConsoleService::Write("Неправильный ввод\n");

isCorrectInput = false;

continue;

}

isCorrectInput = true;

}

return month;

}

int GetInputDay()

{

string dd;

int day;

bool isCorrectInput = false;

while (!isCorrectInput)

{

cin >> dd;

regex regular("(\\d{2}||\\d{1})");

try

{

if (!regex\_match(dd, regular))

{

throw 0;

}

day = stoi(dd);

if (day > 31 || day < 1)

{

throw 0;

}

if (year % 4 == 0 && month == 2 && day > 29)

{

throw 0;

}

if (year % 4 != 0 && month == 2 && day > 28)

{

throw 0;

}

}

catch (...)

{

ConsoleService::Write("Неправильный ввод\n");

isCorrectInput = false;

continue;

}

isCorrectInput = true;

}

return day;

}

public:

int GetYear()

{

return year;

}

int GetMonth()

{

return month;

}

int GetDay()

{

return day;

}

void SetDate()

{

int yy, mm, dd;

ConsoleService::Write("Введите год: ");

year = GetInputYear();

ConsoleService::Write("Введите месяц: ");

month = GetInputMonth();

ConsoleService::Write("Введите день: ");

day = GetInputDay();

convDate = year \* 365 + month \* 31 + day;

}

string GetFullDate()

{

return to\_string(year) + ":" + to\_string(month) + ":" + to\_string(day);

}

long long GetConvDate()

{

return convDate;

}

};

Employee.h

#pragma once

#pragma warning(disable : 4996)

#include <iostream>

#include <string.h>

#include <regex>

#include "Person.h"

#include "ConsoleService.h"

#include "EmployeeType.h"

using namespace std;

class Employee : public Person

{

private:

char address[128];

char phone[128];

int numberHourPerMonth;

int salary;

EmployeeType employeeType;

static constexpr float INCOME\_TAX = 0.12f;

// static constexpr int FULL\_WORKING\_HOURS = 144;

int GetInput()

{

string numberString;

int number;

bool isCorrectNumber = false;

while (!isCorrectNumber)

{

cin >> numberString;

try

{

number = stoi(numberString);

if (number < 0)

{

throw 0;

}

}

catch (...)

{

ConsoleService::Write("ERROR\n");

isCorrectNumber = false;

continue;

}

isCorrectNumber = true;

}

return number;

}

string GetInputPhoneNumber()

{

string phoneNumber;

bool isCorrectNumber = false;

while (!isCorrectNumber)

{

cin >> phoneNumber;

regex regular("^(\\+375)(29|25|33|44)(\\d{7})$");

try

{

if (!regex\_match(phoneNumber, regular))

{

throw 0;

}

}

catch (...)

{

ConsoleService::Write("Неверный ввод\n");

isCorrectNumber = false;

continue;

}

isCorrectNumber = true;

}

return phoneNumber;

}

public:

EmployeeType GetEmployeeType() {

return employeeType;

}

string GetAddress()

{

return address;

}

string GetPhone()

{

return phone;

}

int GetNumberHourPerMonth()

{

return numberHourPerMonth;

}

int GetSalary()

{

return salary;

}

void SetEmployee()

{

ConsoleService::Write("Введите должность работника (цифру): \n0. Директор\n1. Бухгалтер\n2. Наемный рабочий\n3. Уборщик\n");

employeeType = (EmployeeType)GetInputEmployeeType();

cin.ignore();

ConsoleService::Write("Введите имя работника: ");

cin.getline(firstName, 128);

ConsoleService::Write("Введите фамилию работника: ");

cin.getline(lastName, 128);

ConsoleService::Write("Введите отчество работника: ");

cin.getline(middleName, 128);

ConsoleService::Write("Введите адрес работника: ");

cin.getline(address,128);

ConsoleService::Write("Введите телефон работника (+375 29|44|25|33 7цифр): ");

strcpy(phone, GetInputPhoneNumber().c\_str());

ConsoleService::Write("Введите отработанные часы: ");

numberHourPerMonth = GetInput();

ConsoleService::Write("Введите зарплату работника: ");

int salaryClear = GetInput();

salary = salaryClear - salaryClear \* INCOME\_TAX;

}

private:

int GetInputEmployeeType()

{

string numberString;

int number;

bool isCorrectNumber = false;

while (!isCorrectNumber)

{

cin >> numberString;

try

{

number = stoi(numberString);

if (number < 0 || number > 3) {

throw 0;

}

}

catch (...)

{

ConsoleService::Write("ERROR\n");

isCorrectNumber = false;

continue;

}

isCorrectNumber = true;

}

return number;

}

};

Person.h

#pragma once

#pragma warning(disable : 4996)

#include <iostream>

#include <string.h>

using namespace std;

class Person

{

protected:

char firstName[128];

char lastName[128];

char middleName[128];

public:

string GetFirtsName()

{

return firstName;

}

string GetLastName()

{

return lastName;

}

string GetMiddleName()

{

return middleName;

}

};

EmployeeType.h

#pragma once

enum class EmployeeType

{

Director,

Accountant,

Labourer,

Cleaner

};

ConsoleService.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

class ConsoleService

{

public:

static void Write(string text)

{

cout << text;

}

};

FileService.h

#pragma once

#include <iostream>

#include <string>

#include "Storage.h"

#include <fstream>

#include "Account.h"

using namespace std;

class FileService

{

private:

template<typename T1, typename T2>

static void Write(string file, T1 object, T2 collection)

{

ofstream out(file, ofstream::out);

if (out.is\_open())

{

for (const T1& object : collection)

{

out.write((char\*)&object, sizeof(T1));

}

}

out.close();

}

public:

static void WriteFiles()

{

Account account;

Write("files\\accounts.bin", account, Storage::accounts);

}

static void ReadFiles()

{

Storage::accounts.clear();

ifstream in("files\\accounts.bin", ios::app);

if (in.is\_open())

{

Account account;

while (in.read((char\*)&account, sizeof(account)))

{

Storage::accounts.push\_back(account);

}

}

in.close();

}

};

SortService.h

#pragma once

#include "Account.h"

#include <algorithm>

#include "ViewService.h"

class SortService

{

public:

void SortAccountsDescending()

{

vector<Account> accounts = Storage::accounts;

sort(accounts.begin(), accounts.end(), [](Account a, Account b)

{

return a.GetDate().GetConvDate() < b.GetDate().GetConvDate();

});

ViewService view;

view.ShowAccounts(accounts);

}

void SortAccountsAscending()

{

vector<Account> accounts = Storage::accounts;

sort(accounts.begin(), accounts.end(), [](Account a, Account b)

{

return a.GetDate().GetConvDate() > b.GetDate().GetConvDate();

});

ViewService view;

view.ShowAccounts(accounts);

}

};

StorageService.h

#pragma once

#include "Account.h"

#include "Storage.h"

#include "ConsoleService.h"

using namespace std;

class StorageService

{

private:

int GetInput()

{

string menuNumberString;

int menuNumber;

bool isCorrectNumber = false;

while (!isCorrectNumber)

{

ConsoleService::Write("Введите id:\n");

cin >> menuNumberString;

try

{

menuNumber = stoi(menuNumberString);

}

catch (...)

{

ConsoleService::Write("ERROR\n");

isCorrectNumber = false;

continue;

}

isCorrectNumber = true;

}

return menuNumber;

}

public:

void AddAccount()

{

Employee employee;

employee.SetEmployee();

Date date;

date.SetDate();

Account account;

account.SetAccount(date, employee);

Storage::accounts.push\_back(account);

}

void DeleteAccount()

{

int id = GetInput();

if (id < 0 || id >= Storage::accounts.size())

{

ConsoleService::Write("id не существует\n");

return;

}

Storage::accounts.erase(Storage::accounts.begin() + id);

}

};

ViewService.h

#pragma once

#include "Storage.h"

#include "ConsoleService.h"

#include "EmployeeType.h"

using namespace std;

class ViewService

{

public:

void ShowAccounts()

{

ConsoleService::Write("Accounts\n");

for (int i = 0; i < Storage::accounts.size(); i++)

{

ConsoleService::Write("Id: " + to\_string(i) + "\n");

ConsoleService::Write("Должность: ");

switch (Storage::accounts[i].GetEmployee().GetEmployeeType()) {

case EmployeeType::Director: {

ConsoleService::Write("Директор");

break;

}

case EmployeeType::Accountant: {

ConsoleService::Write("Бухгалтер");

break;

}

case EmployeeType::Labourer: {

ConsoleService::Write("Наемный рабочий");

break;

}

case EmployeeType::Cleaner: {

ConsoleService::Write("Уборщик");

break;

}

}

ConsoleService::Write("Имя сотрудника: " + Storage::accounts[i].GetEmployee().GetFirtsName() + "\n");

ConsoleService::Write("Фамилия сотрудника: " + Storage::accounts[i].GetEmployee().GetLastName() + "\n");

ConsoleService::Write("Отчество сотрудника: " + Storage::accounts[i].GetEmployee().GetMiddleName() + "\n");

ConsoleService::Write("Адрес сотрудника: " + Storage::accounts[i].GetEmployee().GetAddress() + "\n");

ConsoleService::Write("Зарплата сотрудника: " + to\_string(Storage::accounts[i].GetEmployee().GetSalary()) + "\n");

ConsoleService::Write("Отработанные часы сотрудника: " + to\_string(Storage::accounts[i].GetEmployee().GetNumberHourPerMonth()) + "\n");

ConsoleService::Write("Телефон сотрудника: " + Storage::accounts[i].GetEmployee().GetPhone() + "\n");

ConsoleService::Write("Дата записи: " + Storage::accounts[i].GetDate().GetFullDate() + "\n");

ConsoleService::Write("\n");

}

}

void ShowAccountsByEmployeeFirstNameAndLastName()

{

string lastName, firstName;

ConsoleService::Write("Введите имя сотрудника\n");

cin >> firstName;

ConsoleService::Write("Введите фамилию сотрудника: \n");

cin >> lastName;

bool isFind = false;

ConsoleService::Write("Accounts by fn and ln\n");

for (int i = 0; i < Storage::accounts.size(); i++)

{

if (firstName == Storage::accounts[i].GetEmployee().GetFirtsName() && lastName == Storage::accounts[i].GetEmployee().GetLastName())

{

ConsoleService::Write("Id: " + to\_string(i) + "\n");

ConsoleService::Write("Должность: ");

switch (Storage::accounts[i].GetEmployee().GetEmployeeType()) {

case EmployeeType::Director: {

ConsoleService::Write("Директор");

break;

}

case EmployeeType::Accountant: {

ConsoleService::Write("Бухгалтер");

break;

}

case EmployeeType::Labourer: {

ConsoleService::Write("Наемный рабочий");

break;

}

case EmployeeType::Cleaner: {

ConsoleService::Write("Уборщик");

break;

}

}

ConsoleService::Write("Имя сотрудника: " + Storage::accounts[i].GetEmployee().GetFirtsName() + "\n");

ConsoleService::Write("Фамилия сотрудника: " + Storage::accounts[i].GetEmployee().GetLastName() + "\n");

ConsoleService::Write("Отчество сотрудника: " + Storage::accounts[i].GetEmployee().GetMiddleName() + "\n");

ConsoleService::Write("Адрес сотрудника: " + Storage::accounts[i].GetEmployee().GetAddress() + "\n");

ConsoleService::Write("Зарплата сотрудника: " + to\_string(Storage::accounts[i].GetEmployee().GetSalary()) + "\n");

ConsoleService::Write("Отработанные часы сотрудника: " + to\_string(Storage::accounts[i].GetEmployee().GetNumberHourPerMonth()) + "\n");

ConsoleService::Write("Телефон сотрудника: " + Storage::accounts[i].GetEmployee().GetPhone() + "\n");

ConsoleService::Write("Дата записи: " + Storage::accounts[i].GetDate().GetFullDate() + "\n");

ConsoleService::Write("\n");

isFind = true;

}

}

if (isFind == false)

{

ConsoleService::Write("Такого сотрудника нет\n");

}

}

void ShowAccountsByDate()

{

int yy, mm, dd;

ConsoleService::Write("Записи сотрудников за определенную дату\n");

ConsoleService::Write("Введите дату записи: \n");

ConsoleService::Write("Введите год\n");

cin >> yy;

ConsoleService::Write("Введите месяц\n");

cin >> mm;

ConsoleService::Write("Введите день\n");

cin >> dd;

bool isFind = false;

ConsoleService::Write("Accounts by fn and ln\n");

for (int i = 0; i < Storage::accounts.size(); i++)

{

if (Storage::accounts[i].GetDate().GetYear() == yy && Storage::accounts[i].GetDate().GetMonth() == mm

&& Storage::accounts[i].GetDate().GetDay() == dd)

{

ConsoleService::Write("Id: " + to\_string(i) + "\n");

ConsoleService::Write("Должность: ");

switch (Storage::accounts[i].GetEmployee().GetEmployeeType()) {

case EmployeeType::Director: {

ConsoleService::Write("Директор\n");

break;

}

case EmployeeType::Accountant: {

ConsoleService::Write("Бухгалтер\n");

break;

}

case EmployeeType::Labourer: {

ConsoleService::Write("Наемный рабочий\n");

break;

}

case EmployeeType::Cleaner: {

ConsoleService::Write("Уборщик\n");

break;

}

}

ConsoleService::Write("Имя сотрудника: " + Storage::accounts[i].GetEmployee().GetFirtsName() + "\n");

ConsoleService::Write("Фамилия сотрудника: " + Storage::accounts[i].GetEmployee().GetLastName() + "\n");

ConsoleService::Write("Отчество сотрудника: " + Storage::accounts[i].GetEmployee().GetMiddleName() + "\n");

ConsoleService::Write("Адрес сотрудника: " + Storage::accounts[i].GetEmployee().GetAddress() + "\n");

ConsoleService::Write("Зарплата сотрудника: " + to\_string(Storage::accounts[i].GetEmployee().GetSalary()) + "\n");

ConsoleService::Write("Отработанные часы сотрудника: " + to\_string(Storage::accounts[i].GetEmployee().GetNumberHourPerMonth()) + "\n");

ConsoleService::Write("Телефон сотрудника: " + Storage::accounts[i].GetEmployee().GetPhone() + "\n");

ConsoleService::Write("Дата записи: " + Storage::accounts[i].GetDate().GetFullDate() + "\n");

ConsoleService::Write("\n");

isFind = true;

}

}

if (isFind == false)

{

ConsoleService::Write("Такого сотрудника нет\n");

}

}

void ShowAccounts(vector<Account> accounts)

{

ConsoleService::Write("Accounts\n");

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

{

ConsoleService::Write("Id: " + to\_string(i) + "\n");

ConsoleService::Write("Должность: ");

switch (Storage::accounts[i].GetEmployee().GetEmployeeType()) {

case EmployeeType::Director: {

ConsoleService::Write("Директор");

break;

}

case EmployeeType::Accountant: {

ConsoleService::Write("Бухгалтер");

break;

}

case EmployeeType::Labourer: {

ConsoleService::Write("Наемный рабочий");

break;

}

case EmployeeType::Cleaner: {

ConsoleService::Write("Уборщик");

break;

}

}

ConsoleService::Write("Имя сотрудника: " + accounts[i].GetEmployee().GetFirtsName() + "\n");

ConsoleService::Write("Фамилия сотрудника: " + accounts[i].GetEmployee().GetLastName() + "\n");

ConsoleService::Write("Отчество сотрудника: " + accounts[i].GetEmployee().GetMiddleName() + "\n");

ConsoleService::Write("Адрес сотрудника: " + accounts[i].GetEmployee().GetAddress() + "\n");

ConsoleService::Write("Зарплата сотрудника: " + to\_string(accounts[i].GetEmployee().GetSalary()) + "\n");

ConsoleService::Write("Отработанные часы сотрудника: " + to\_string(accounts[i].GetEmployee().GetNumberHourPerMonth()) + "\n");

ConsoleService::Write("Телефон сотрудника: " + accounts[i].GetEmployee().GetPhone() + "\n");

ConsoleService::Write("Дата записи: " + accounts[i].GetDate().GetFullDate() + "\n");

ConsoleService::Write("\n");

}

}

};

Storage.h

#pragma once

#include <iostream>

#include "Account.h"

#include <vector>

using namespace std;

class Storage

{

public:

static vector<Account> accounts;

};

CourseWork.cpp

#include <iostream>

#include "Menu.h"

#include <windows.h>

#include <conio.h>

using namespace std;

int main()

{

setlocale(LC\_ALL, "RUS");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Menu menu;

menu.MainMenu();

}

Menu.h

#pragma once

#include <iostream>

#include <string>

#include "StorageService.h"

#include "ViewService.h"

#include "FileService.h"

#include "SortService.h"

#include "ConsoleService.h"

using namespace std;

class Menu

{

private:

ViewService viewservice;

StorageService storageService;

FileService fileService;

SortService sortService;

public:

void MainMenu()

{

while (true)

{

ConsoleService::Write("1. Добавить запись\n");

ConsoleService::Write("2. Показать записи\n");

ConsoleService::Write("3. Прочитать файл\n");

ConsoleService::Write("4. Сортировать записи\n");

ConsoleService::Write("5. Удалить запись\n");

ConsoleService::Write("6. Сохранить файл\n");

ConsoleService::Write("7. Выход\n");

int menuNumber = GetInput();

system("cls");

switch (menuNumber)

{

case 1: {

storageService.AddAccount();

break;

}

case 2: {

PrintMenu();

break;

}

case 3: {

ConsoleService::Write("Файл считан\n");

fileService.ReadFiles();

break;

}

case 4: {

SortMenu();

break;

}

case 5: {

viewservice.ShowAccounts();

storageService.DeleteAccount();

break;

}

case 6: {

ConsoleService::Write("Файл сохранен\n");

fileService.WriteFiles();

break;

}

case 7: {

return;

}

default:

ConsoleService::Write("ERROR\n");

break;

}

}

}

private:

void PrintMenu()

{

bool isOpenMenu = true;

while (isOpenMenu)

{

ConsoleService::Write("1. Вывести все записи\n");

ConsoleService::Write("2. Вывести записи по дате\n");

ConsoleService::Write("3. Вывести записи по фамилии и имени сотрудника\n");

ConsoleService::Write("4. Назад\n");

int menuNumber = GetInput();

system("cls");

switch (menuNumber)

{

case 1: {

viewservice.ShowAccounts();

break;

}

case 2: {

viewservice.ShowAccountsByDate();

break;

}

case 3: {

viewservice.ShowAccountsByEmployeeFirstNameAndLastName();

break;

}

case 4: {

return;

}

default:

ConsoleService::Write("ERROR\n");

break;

}

}

}

void SortMenu()

{

bool isOpenMenu = true;

while (isOpenMenu)

{

ConsoleService::Write("1. Сортировать записи по дате по возрастанию\n");

ConsoleService::Write("2. Сортировать записи по дате по убыванию\n");

ConsoleService::Write("3. Назад\n");

int menuNumber = GetInput();

system("cls");

switch (menuNumber)

{

case 1: {

sortService.SortAccountsAscending();

break;

}

case 2: {

sortService.SortAccountsDescending();

break;

}

case 3: {

return;

}

default:

ConsoleService::Write("ERROR\n");

break;

}

}

}

int GetInput()

{

string menuNumberString;

int menuNumber;

bool isCorrectNumber = false;

while (!isCorrectNumber)

{

ConsoleService::Write("Введите номер меню:\n");

cin >> menuNumberString;

try

{

menuNumber = stoi(menuNumberString);

}

catch (...)

{

ConsoleService::Write("ERROR.Введите номер из меню\n");

isCorrectNumber = false;

continue;

}

isCorrectNumber = true;

}

return menuNumber;

}

};