Содержание

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | 4 |  | 3 |
| Введение | 3 | 5 |  |  |
| 1 Описание предметной области | 4 | 5 |  | 4 |
| 1.1 Постановка задачи | 4 | 5 |  | 5 |
| 1.2 Составление математических (вербальных моделей) | 5 | 11 |  | 4 |
| 2 Проектирование программного модуля | 6 | 11 |  |  |
| 2.1 Составление перечня функциональности (User Stories) | 6 | 15 |  |  |
| 2.2 Составление сценариев взаимодействия (краткого и подробного) | 6 | 15 |  |  |
| 3 Разработка документации программного модуля | 9 | 19 |  |  |
| 3.1 Разработка диаграммы классов | 9 | 19 |  |  |
| 3.2 Разработка диаграммы деятельности  3.3 Разработка основных алгоритмов | 10  11 | 20 |  |  |
| 4 Разработка программного модуля | 13 | 21 |  |  |
| 4.1 Выбор и обоснование технологий, используемых в программном модуле | 13 | 21 |  |  |
| 4.2 Настройка окружения и репозитория кода | 13 | 26 |  |  |
| 4.3 Описание используемых структур данных | 14 | 33 |  |  |
| 5 Тестирование программного модуля | 17 |  |  |  |
| [Заключение](#_Toc10412337) |  |  |  |  |
| [Список литературы](#_Toc10412338) |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  | 57 |  |  |
|  |  | 58 |  |  |
|  |  | 60 |  |  |
|  |  | 61 |  |  |
|  |  | 62 |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# Введение

Целью курсовой работы является создание программного обеспечения для расчета заработной платы сотрудников предприятия

Задачи курсовой работы:

- изучить предметную область и установить требования к программному продукту;

- спроектировать программный продукт;

- разработать программный продукт;

- протестировать работу программного продукта.

В результате выполнения курсовой работы должно быть разработано полностью рабочее приложение выводить всю базу данных сотрудников предприятия на экран монитора, добавлять новые записи в базу данных, сортировать базу данных по часам работы, удалять записи из базы данных, сохранять базу данных в файл, выводить список записей с заработной платой после вычета налогов, редактировать записи базы данных, выходить из программы.

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

1 Описание предметной области

* 1. Постановка задачи

Целью данной курсовой работы является разработка программы на языке С++ для создания архива сотрудников для расчета заработной платы сотрудников предприятия.

Данные программы должны храниться в файлах и базе данных. Каждая логически завершенная задача программы должна быть реализована в виде метода. Разрабатываемый интерфейс должен быть понятным и защищенным от случайных ошибок, вылетов и ввода некорректных данных. Программа должна содержать не менее двух форм и иметь раздел «Помощь». Ввод исходных данных должен осуществляться либо с клавиатуры, либо из файла, либо из базы данных (по выбору пользователя).

В программе будет возможность ввода сотрудника, работа с базой данных: редактирование, добавление, удаление данных, показ сотрудников, сортировка, расчет заработной платы после вычета налога.

* 1. Составление математических (вербальных) моделей

Вербальная модель –информационная модель в мысленной или разговорной форме.

Знаковая модель – информационная модель, выраженная специальными знаками, т. е. средствами любого формального языка.

К информационным моделям можно отнести вербальные (от лат.«verbalize» — устный) модели, полученные в результате раздумий, умозаключений. Они могут так и остаться мысленными или быть выражены словесно. Примером такой модели может стать наше поведение при переходе улицы. Человек анализирует ситуацию на дороге (что показывает светофор, как далеко находятся машины, с какой скоростью они движутся и т. п.) и вырабатывает свою модель поведения. Если ситуация смоделирована правильно, то переход будет безопасным, если нет, то может произойти авария. К таким моделям можно отнести и идею, возникшую у изобретателя, и музыкальную тему, промелькнувшую в голове композитора, и рифму, прозвучавшую пока еще в сознании поэта.

Знаковые модели – это: рисунки, тексты, графики и схемы. Вербальные и знаковые модели, как правило, взаимосвязаны. Мысленный образ, родившийся в мозгу человека, может быть облечен в знаковую форму. И наоборот, знаковая модель помогает сформировать в сознании верный мысленный образ.

По форме представления можно выделить следующие виды информационных моделей:

* геометрические модели - графические формы и объемные конструкции;
* словесные модели - устные и письменные описания с использованием иллюстраций;
* математические модели - математические формулы, отображающие связь различных параметров объекта или процесса;
* структурные модели - схемы, графики, таблицы и т. п.;
* логические модели - модели, в которых представлены различные варианты выбора действий на основе умозаключений и анализа условий;
* специальные модели - ноты, химические формулы и т. п.;
* компьютерные и некомпьютерные модели.

На рисунке 1.1 представлена математическая модель программно-информационной системы, на которой можно увидеть действия, доступные пользователю.



Рисунок 1.1 –Математическая модель программы

2 Проектирование программного модуля

* 1. Составление перечня функциональности

Разрабатываемая программа должна обеспечивать возможность выполнения перечисленных ниже функций:

* ввод исходных данных;
* запуск процесса вычислений;
* просмотр результатов вычислений;
* показ результатов вычислений;
* запись результатов вычислений;
* показ подсказок пользователю;
* добавление данных;
* удаление данных;
* изменение данных.

2.2 Составление сценария взаимодействия

Полный сценарий взаимодействия:

Программа предлагает пользователю некоторые манипуляции с данными. Пользователь может добавлять, изменять, удалять и просматривать данные при помощи специальных элементов управления.

1. Пользователь попадает в главное меню, где ему даётся выбор действий. Пользователь может ввести сотрудника и узнать информацию о нужном сотруднике. Может добавить, изменить или удалить информацию о сотруднике, а также получить информацию про эти функции.
2. При нажатии кнопки “Safe file” программа создаёт новый файл с названием “Storage.txt”, где данные записываются в этот файл, который можно прочесть или использовать при новом запуске программы.
3. При нажатии копки “Show data” открывается окно информации, где пользователь может узнать данные обо всех сотрудниках.
4. При нажатии копки “Add employee” открывается окно, где пользователь может добавить сотрудника предприятия.
5. При нажатии копки “Delete employee” открывается окно, где пользователь может удалить запись о желаемом сотруднике.
6. При нажатии кнопки “Edit employee” открывается окно изменения данных, в котором можно изменить данные о любом сотруднике.
7. При нажатии кнопки “Sort by hours” открывается окно, где пользователь может выбрать сортировку записей по часам работы.
8. При нажатии кнопки “Calculate salary” открывается окно, в котором показывается информация о сотрудниках после вычета подоходного налога.
9. При нажатии кнопки “Exit” программа закрывается.

При запуске программы пользователь увидит мини-инструкцию и главное меню программы, в котором можно добавлять новых сотрудников, смотреть уже введённых, вносить изменения, удалять сотрудников, сохранять в файл, проводить сортировку и проводить расчет заработной платы после вычета подоходного налога.

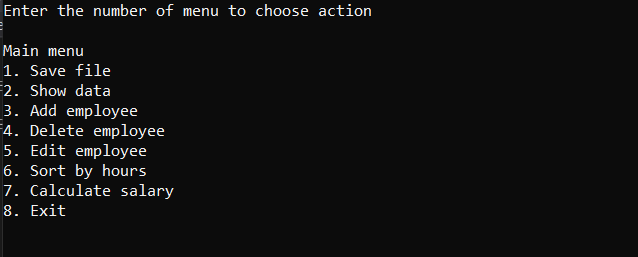


Рисунок 2.1 – Окно с мини-инструкцией и главным меню

При выборе пункта “Show data” пользователь получит информацию обо всех записанных в программу сотрудниках.

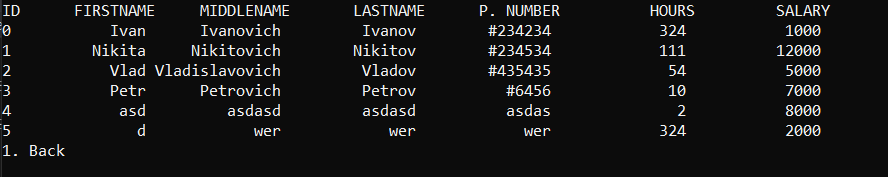


Рисунок 2.2 – Окно информации обо всех сотрудниках.

При вызове пункта “ Edit employee”, пользователь попадёт в под-меню, в котором он может либо вернуться в меню, либо внести сами изменения. При выборе пункта внесения изменения, пользователю выводятся списки сотрудников, в которое он должен ввести для изменения в нём определённого пункта.

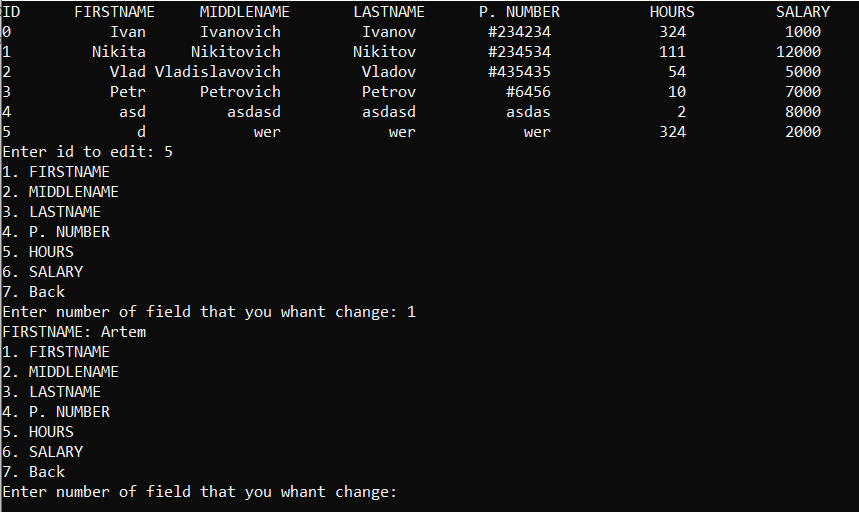


Рисунок 2.3 – Окно внесения изменений

При выборе пункта “Save file”, программа создаёт файл Storage.txt,в который записывает все элементы класса. При успешной перезаписи выводится запись “Saving was successful!”.

При выборе пункта “Sort by hours” пользователь попадает в подменю с тремя пунктами: “Ascending sort”, “Descending sort”, и “Back”. При выборе первого пункта, пользователь попадает в окно с отсортированными часами работы по возрастанию. При выборе второго пункта, пользователь попадает в окно с отсортированными часами работы по убыванию. При выборе третьего пункта, пользователь попадает в главное меню программы.

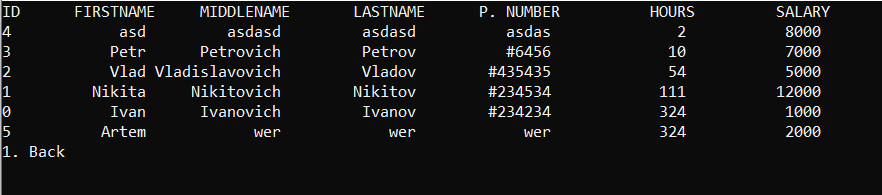


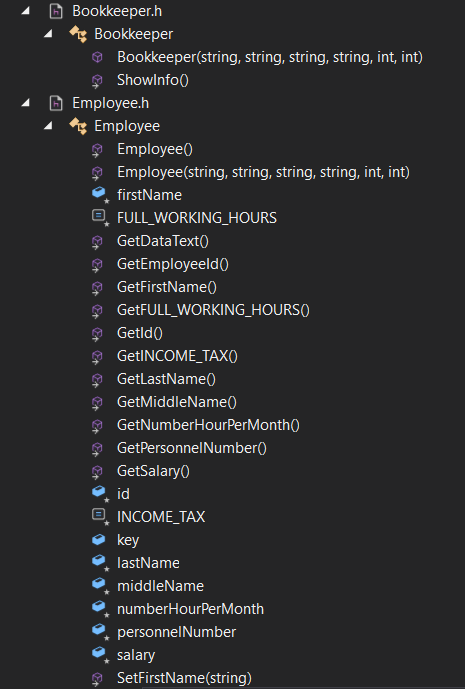
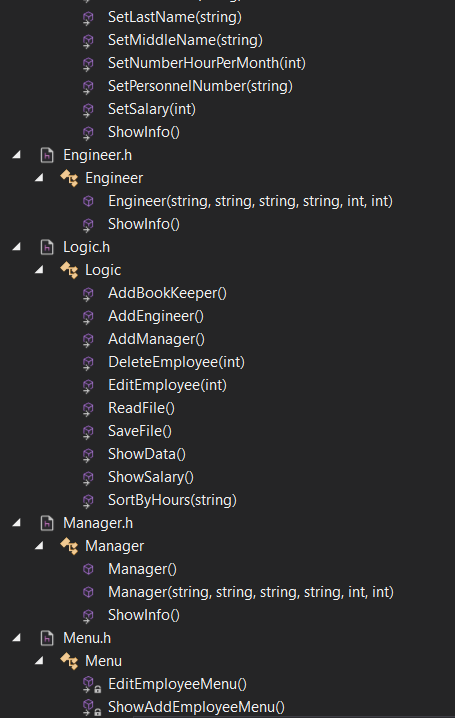
Рисунок 2.4 – Окно сортировки по возрастанию

При выборе пункта “Закончить работу” программа закрывается.

3 Разработка документации программного модуля

* 1. Разработка диаграммы классов

Диаграмма классов приложения представлены на рисунке 3.1.

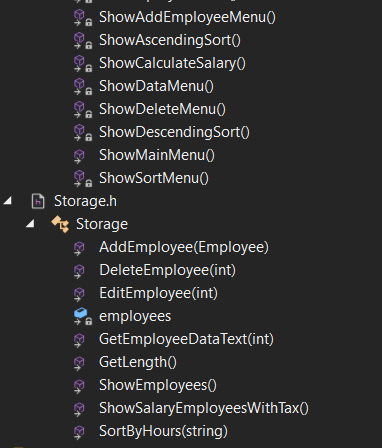


Рисунок 3.1 - Диаграмма классов

* 1. Разработка диаграммы деятельности

Под диаграммой деятельности понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов - вложенных видов деятельности и отдельных действий, соединённых между собой.

Диаграмма деятельности (рисунок 3.2) используется при моделировании бизнес-процессов, технологических процессов, последовательных и параллельных вычислений.

* Диаграммы деятельности состоят из ограниченного количества фигур, соединённых стрелками. Основные фигуры:
* Прямоугольники с закруглениями – действия
* Ромбы - решения
* Чёрный круг - начало процесса (начальный узел)



Рисунок 3.2 – Диаграмма деятельности

* 1. Разработка основных алгоритмов

Алгоритм - конечная совокупность точно заданных правил решения произвольного класса задач или набор инструкций, описывающих порядок действий исполнителя для решения некоторой задачи.

То есть алгоритм – это чёткое указание исполнителю алгоритма выполнить определённую последовательность действий для решения поставленной задачи и получения результата.

Основные особенности алгоритмов:

* Наличие ввода исходных данных
* Наличие вывода результата выполнения алгоритма, поскольку цель выполнения алгоритма - получение результата, имеющего вполне определённое отношение к исходным данным.
* Алгоритм должен иметь дискретную структуру, т е. алгоритм представляется в виде последовательности шагов.
* Однозначность – каждый шаг алгоритма должен быть чётко определён.
* Конечность – исполнение алгоритма должно закончиться за конечное число шагов
* Корректность – алгоритм должен задавать правильное решение задачи
* Массовость – алгоритм разрабатывается для решения некоторого класса задач, различающихся исходными данными.
* Эффективность – алгоритм должен выполняться за разумное конечное время.

Блок-схема алгоритма – графическое изображение алгоритма в виде связанных между собой с помощью стрелок (линии перехода) и блоков - графических символов, каждый из которых соответствует одному шагу алгоритма. Внутри блока даётся описание соответствующего действия.

На рисунке 3.3 представлен основной алгоритм программы.

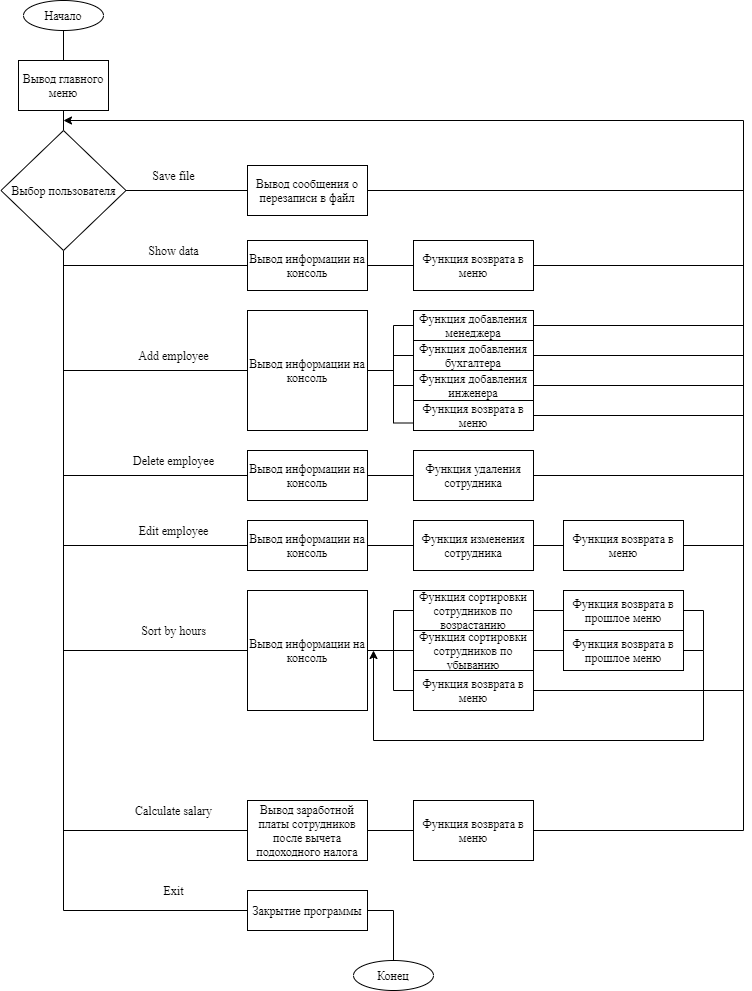


Рисунок 3.3 – Основной алгоритм системы

4 Разработка программного модуля

* 1. Выбор и обоснование технологий, используемых в программном модуле

В качестве платформы для разработки программы используется .NET Microsoft Visual Studio 2019.

Возможности Visual Studio 2019:

1. Разработка – быстрые навигация, написание и исправление кода.
2. Отладка – простая отладка, профилирование и диагностика кода.
3. Тестирование -комплексные инструменты тестирования помогают писать высококачественный код.
4. Сотрудничество – используется система управления версиями для гибкой и эффективной совместной работы.
5. Расширение – тысячи расширений позволяют настроить IDE под себя.

В качестве языка разработки был выбран язык C++.

Язык C++ и связанную с ним среду .NET Framework можно без преувеличения назвать самой значительной из предлагаемых в настоящее время технологий для разработчиков. Среда .NET является такой средой, которая была создана для того, чтобы в ней можно было разрабатывать практически любое приложение для запуска в Windows, а C# является языком программирования, который был специально создан для использования в .NET Framework.

С++ - относительно новый язык программирования, который характеризуется двумя следующими преимуществами:

1. C++ спроектирован и разработан специально для применения с Microsoft .NET Framework (развитой платформой разработки развертывания и выполнения распределённых приложений).
2. C++ - язык, основанный на современной объектно-ориентированной методологии проектирования, при разработке которого специалисты из Microsoft опирались на опыт создания подобных языков, построенных в соответствии с предложенными около 20 лет назад объектно-ориентированными принципами.
   1. Настройка окружения и репозитория кода

Система управления версиями (репозиторий кода) (от англ. Version Control System, VCS или Revision Control System) — программное обеспечение для облегчения работы с изменяющейся информацией. Она позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Ситуация, в которой электронный документ за время своего существования претерпевает ряд изменений, достаточно типична. При этом часто бывает важно иметь не только последнюю версию, но и несколько предыдущих. В простейшем случае можно просто хранить несколько вариантов документа, нумеруя их соответствующим образом. Такой способ неэффективен (приходится хранить несколько практически идентичных копий), требует повышенного внимания и дисциплины и часто ведёт к ошибкам, поэтому были разработаны средства для автоматизации этой работы.

Рассматриваемые системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы.

* 1. Описание используемых структур данных

Описание класса выглядит как в коде ниже:

protected:

int id;

string firstName;

string middleName;

string lastName;

string personnelNumber;

int numberHourPerMonth;

int salary;

static constexpr float INCOME\_TAX = 0.12f;

static constexpr int FULL\_WORKING\_HOURS = 144;

public:

string GetDataText();

int key = numberHourPerMonth;

virtual void ShowInfo();

int GetEmployeeId();

string GetFirstName();

string GetMiddleName();

string GetLastName();

string GetPersonnelNumber();

int GetNumberHourPerMonth();

int GetSalary();

void SetFirstName(string firstName);

void SetMiddleName(string middleName);

void SetLastName(string lastName);

void SetPersonnelNumber(string personalNumber);

void SetNumberHourPerMonth(int numberHourPerMonth);

void SetSalary(int salary);

float GetINCOME\_TAX();

int GetFULL\_WORKING\_HOURS();

int GetId();

Employee(string firstName, string middleName, string lastName, string personnelNumber, int numberHourPerMonth, int salary);

Employee();

Имеются protected- и public-операторы, в protected занесены переменные, а в public — все функции, которые пригодятся для использования в программе. Геттеры и сеттеры необходимы для чтения и записи переменных в класс и из класса. Констуктор Employee() приваивает всем элементам пустое значение по умолчанию.

Employee::Employee() {

static int count = 0;

id = count++;

this->firstName = "Undefined";

this->middleName = "Undefined";

this->lastName = "Undefined";

this->personnelNumber = "Undefined";

this->numberHourPerMonth = 0;

this->salary = 0;

}

Программа также использует свою библиотеку Storage.h, в которой описаны функции:

private:

static vector<Employee> employees;

public:

static void ShowSalaryEmployeesWithTax();

static string GetEmployeeDataText(int i);

static int GetLength();

static void AddEmployee(Employee employee);

static void DeleteEmployee(int employeeId);

static void ShowEmployees();

static void SortByHours(string typeSort);

static void EditEmployee(int id);

Функция static void ShowSalaryEmployeesWithTax(); предоставляет пользователю информацию о сотрудниках после вычета подоходного налога. static void AddEmployee(Employee employee); предоставляет возможность добавления нового сотрудника. Функция static void EditEmployee(int id); позволяет пользователю внести изменения в уже имеющегося сотрудника в базе данных. Функция static void SortByHours(string typeSort); позволяет пользователю вывести данные с помощью определённого вида сортировки. Функция static void DeleteEmployee(int employeeId); позволяет удалить сотрудника из базы данных. Функция static void ShowEmployees(); предоставляет пользователю информацию о всех сотрудниках.

Программный код представлен в Приложении А.

5 Тестирование программного модуля

При запуске программы открывается главное меню программы. Пользователь может выбрать один из пунктов программы и сразу приступить к работе.

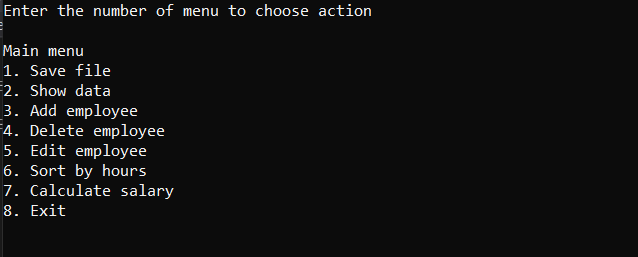


Рисунок 5.1 — Главное меню

При выборе пункта 1, сохраняется информация о сотрудниках в файл Storage.

При выборе пункта 2, выводятся все имеющиеся сотрудники (записанные в файле Storage.txt), после чего можно вернуться в главное меню:

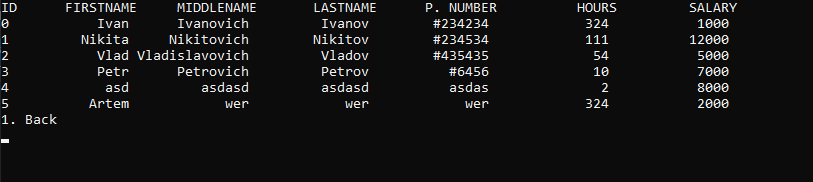


Рисунок 5.2 — Вывод всех элементов

При выборе пункта 3, программа переносит пользователя в меню с выборами из четырех пунктов:

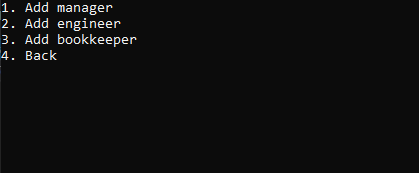


Рисунок 5.3 — Элементы пункта 3

Выбрав в этом меню пункт 1, пользователя сразу же попросят внести все новые элементы. К сожалению, проверка на правильность не предусмотрена.

Выбрав в меню пункт 4, пользователь вновь окажется в главном меню:

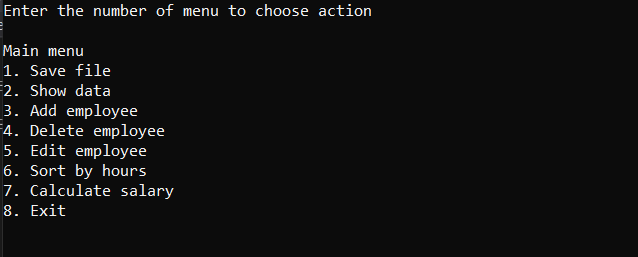


Рисунок 5.5 — Главное меню

При выборе пункта 4, пользователю предлагают выбрать из списка сотрудника, которого требуется удалить.

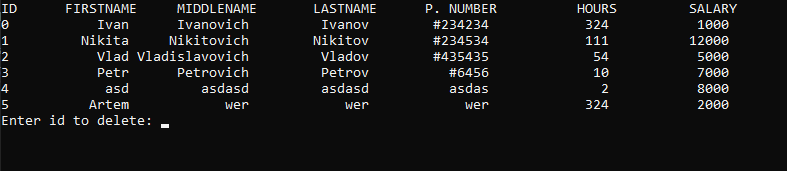


Рисунок 5.6 – Элементы пункта 4

Выбрав пункт 5, пользователь должен выбрать номер из списка сотрудников, о котором надо изменить информацию. Также имеется возможность вернуться в главное меню выбрав 7 пункт.

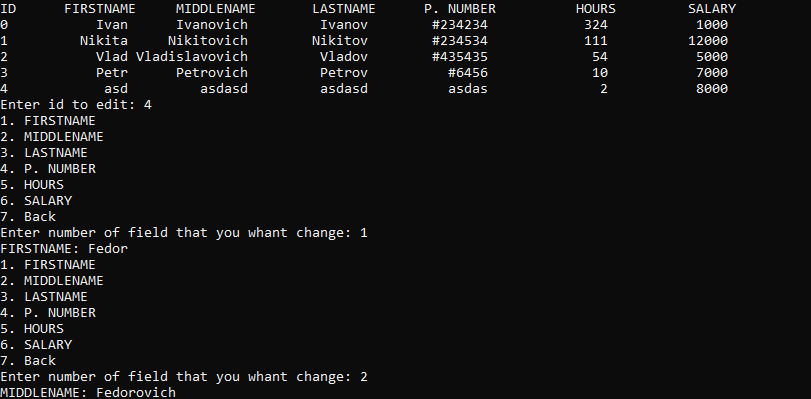


Рисунок 5.7 – Элементы пункта 5

При выборе пункта 6, пользователю предлагают выбрать сортировку по возрастанию или по убыванию. Также можно вернуться в главное меню.

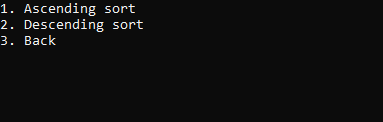


Рисунок 5.8 – Элементы пункта 6

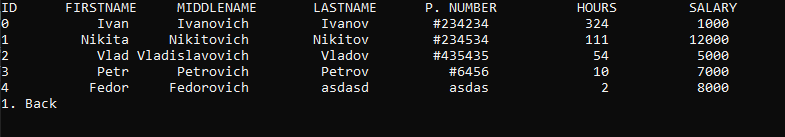


Рисунок 5.9 – Элементы подпункта “Descending Sort”

Выбрав пункт 7, пользователю предоставляется информация о сотрудниках после вычета подоходного налога.

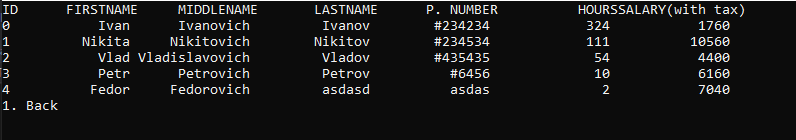


Рисунок 5.10 – Вывод 7 пункта на консоль

При выборе пункта 8, программа завершается.

# Заключение

В ходе проделанной работы, была разработана программа для создания базы данных сотрудников и требуемых расчетов. Тестированием была подтверждена корректная работа программы.

В ходе выполнения курсовой работы были закрепление основы и углублены знания приемов программирования на языке C++. При реализации проекта получены практические навыки на всех этапах создания программного продукта: от постановки задачи до практической реализации, сопровождающейся документацией и инструкциями по его использованию.

В дальнейшем программа может развиваться и усложняться.

Программа предназначена для наглядной демонстрации работы с базой данных.

К достоинствам программного продукта можно отнести понятный для пользователя интерфейс.

# Список литературы

1 Сайт о программировании — <https://metanit.com/>

2 Сайт о программировании на .NET Professorweb.ru [Электронный ресурс] — 2017. — Режим доступа: <https://professorweb.ru/>.

3 Справочный материал MSDN.Microsoft.com [Электронный ресурс] — 2019. — Режим доступа: <https://msdn.microsoft.com/library>.

4 Общедоступная многоязычная универсальная интернет-энциклопедия со свободным контентом [Электронный ресурс] — 2019. — https://ru.wikipedia.org

# Приложение А

Код программы

Bookkeeper.h

#pragma once

#include "Employee.h"

#include <iostream>

#include <string>

using namespace std;

class Bookkeeper : public Employee

{

public:

void ShowInfo() override;

Bookkeeper(string firstName, string middleName, string lastName, string personnelNumber, int numberHourPerMonth, int salary)

: Employee(firstName, middleName, lastName, personnelNumber, numberHourPerMonth, salary) {

}

};

Employee.h

#pragma once

#include <iostream>

#include <string>

#include <iomanip>

using namespace std;

class Employee

{

protected:

int id;

string firstName;

string middleName;

string lastName;

string personnelNumber;

int numberHourPerMonth;

int salary;

static constexpr float INCOME\_TAX = 0.12f;

static constexpr int FULL\_WORKING\_HOURS = 144;

public:

string GetDataText();

int key = numberHourPerMonth;

virtual void ShowInfo();

int GetEmployeeId();

string GetFirstName();

string GetMiddleName();

string GetLastName();

string GetPersonnelNumber();

int GetNumberHourPerMonth();

int GetSalary();

void SetFirstName(string firstName);

void SetMiddleName(string middleName);

void SetLastName(string lastName);

void SetPersonnelNumber(string personalNumber);

void SetNumberHourPerMonth(int numberHourPerMonth);

void SetSalary(int salary);

float GetINCOME\_TAX();

int GetFULL\_WORKING\_HOURS();

int GetId();

Employee(string firstName, string middleName, string lastName, string personnelNumber, int numberHourPerMonth, int salary);

Employee();

};

Engineer.h

#pragma once

#include "Employee.h"

#include <iostream>

#include <string>

using namespace std;

class Engineer : public Employee

{

public:

void ShowInfo() override;

Engineer(string firstName, string middleName, string lastName, string personnelNumber, int numberHourPerMonth, int salary)

: Employee(firstName, middleName, lastName, personnelNumber, numberHourPerMonth, salary) {

}

};

Logic.h

#pragma once

#include <string>

#include <fstream>

#include <iostream>

#include <windows.h>

#include "Employee.h"

#include "Manager.h"

#include "Engineer.h"

#include "Bookkeeper.h"

#include "Storage.h"

using namespace std;

class Logic

{

public:

static void DeleteEmployee(int id);

static void EditEmployee(int id);

static void ShowData();

static void ShowSalary();

static void AddManager();

static void AddEngineer();

static void AddBookKeeper();

static void SortByHours(string typeSort);

static void ReadFile();

static void SaveFile();

};

Manager.h

#pragma once

#include "Employee.h"

#include <iostream>

#include <string>

using namespace std;

class Manager : public Employee

{

public:

void ShowInfo() override;

Manager(string firstName, string middleName, string lastName, string personnelNumber, int numberHourPerMonth,int salary)

: Employee(firstName, middleName, lastName, personnelNumber, numberHourPerMonth, salary) {

}

Manager() {

}

};

Menu.h

#pragma once

#include <iostream>

#include <string>

#include <stdlib.h>

#include "Logic.h"

using namespace std;

class Menu

{

public:

static void ShowMainMenu();

private:

static void ShowDeleteMenu();

static void EditEmployeeMenu();

static void ShowSortMenu();

static void ShowAscendingSort();

static void ShowDescendingSort();

static void ShowCalculateSalary();

static void ShowDataMenu();

static void ShowAddEmployeeMenu();

};

Storage.h

#pragma once

#include <vector>

#include <iomanip>

#include <algorithm>

#include "Employee.h"

#include "Manager.h"

#include "Engineer.h"

#include "Bookkeeper.h"

using namespace std;

class Storage

{

private:

static vector<Employee> employees;

public:

static void ShowSalaryEmployeesWithTax();

static string GetEmployeeDataText(int i);

static int GetLength();

static void AddEmployee(Employee employee);

static void DeleteEmployee(int employeeId);

static void ShowEmployees();

static void SortByHours(string typeSort);

static void EditEmployee(int id);

};

Bookkeeper.cpp

#include "Bookkeeper.h"

void Bookkeeper::ShowInfo() {

cout << id << " " << "Bookkeeper" << " " << firstName << " " << lastName << " " << lastName << " " << numberHourPerMonth << endl;

}

Employee.cpp

#include "Employee.h"

string Employee::GetFirstName() {

return firstName;

}

string Employee::GetMiddleName() {

return middleName;

}

string Employee::GetLastName(){

return lastName;

}

string Employee::GetPersonnelNumber(){

return personnelNumber;

}

int Employee::GetSalary(){

return salary;

}

void Employee::SetFirstName(string firstName)

{

this->firstName = firstName;

}

void Employee::SetMiddleName(string middleName)

{

this->middleName = middleName;

}

void Employee::SetLastName(string lastName)

{

this->lastName = lastName;

}

void Employee::SetPersonnelNumber(string personalNumber)

{

this->personnelNumber = personalNumber;

}

void Employee::SetNumberHourPerMonth(int numberHourPerMonth)

{

this->numberHourPerMonth = numberHourPerMonth;

}

void Employee::SetSalary(int salary)

{

this->salary = salary;

}

float Employee::GetINCOME\_TAX(){

return INCOME\_TAX;

}

int Employee::GetFULL\_WORKING\_HOURS(){

return FULL\_WORKING\_HOURS;

}

string Employee::GetDataText() {

return firstName + " " + middleName + " " + lastName + " " + personnelNumber + " " + to\_string(numberHourPerMonth) + " " + to\_string(salary);

}

int Employee::GetId() {

return id;

}

void Employee::ShowInfo() {

cout << id << setw(15) << firstName << setw(15) << middleName << setw(15) << lastName

<< setw(15) << personnelNumber << setw(15) << numberHourPerMonth << setw(15) << salary << endl;

};

int Employee::GetEmployeeId() {

return id;

}

int Employee::GetNumberHourPerMonth() {

return numberHourPerMonth;

}

Employee::Employee(string firstName, string middleName, string lastName, string personnelNumber, int numberHourPerMonth, int salary) {

static int count = 0;

id = count++;

this->firstName = firstName;

this->middleName = middleName;

this->lastName = lastName;

this->personnelNumber = personnelNumber;

this->numberHourPerMonth = numberHourPerMonth;

this->salary = salary;

}

Employee::Employee() {

static int count = 0;

id = count++;

this->firstName = "Undefined";

this->middleName = "Undefined";

this->lastName = "Undefined";

this->personnelNumber = "Undefined";

this->numberHourPerMonth = 0;

this->salary = 0;

}

Engineer.cpp

#include "Engineer.h"

void Engineer::ShowInfo() {

cout << id << " " << "Engineer" << " " << firstName << " " << lastName << " " << lastName << " " << numberHourPerMonth << endl;

}

Logic.cpp

#include "Logic.h"

void Logic::DeleteEmployee(int id) {

Storage::DeleteEmployee(id);

}

void Logic::ShowData() {

Storage::ShowEmployees();

}

void Logic::ShowSalary() {

Storage::ShowSalaryEmployeesWithTax();

}

void Logic::AddManager() {

string firstName;

string middleName;

string lastName;

string personnelNumber;

int numberHourPerMonth;

int salary;

cout << "Enter first name: ";

cin >> firstName;

cout << "Enter middle name: ";

cin >> middleName;

cout << "Enter last name: ";

cin >> lastName;

cout << "Enter personnel number: ";

cin >> personnelNumber;

cout << "Enter number of work hour per month: ";

cin >> numberHourPerMonth;

cout << "Enter salary: ";

cin >> salary;

Storage::AddEmployee(Manager(firstName, middleName, lastName, personnelNumber, numberHourPerMonth, salary));

}

void Logic::AddEngineer() {

string firstName;

string middleName;

string lastName;

string personnelNumber;

int numberHourPerMonth;

int salary;

cout << "Enter first name: ";

cin >> firstName;

cout << "Enter middle name: ";

cin >> middleName;

cout << "Enter last name: ";

cin >> lastName;

cout << "Enter personnel number: ";

cin >> personnelNumber;

cout << "Enter number of work hour per month: ";

cin >> numberHourPerMonth;

cout << "Enter salary: ";

cin >> salary;

Storage::AddEmployee(Engineer(firstName, middleName, lastName, personnelNumber, numberHourPerMonth, salary));

}

void Logic::AddBookKeeper() {

string firstName;

string middleName;

string lastName;

string personnelNumber;

int numberHourPerMonth;

int salary;

cout << "Enter first name: ";

cin >> firstName;

cout << "Enter middle name: ";

cin >> middleName;

cout << "Enter last name: ";

cin >> lastName;

cout << "Enter personnel number: ";

cin >> personnelNumber;

cout << "Enter number of work hour per month: ";

cin >> numberHourPerMonth;

cout << "Enter salary: ";

cin >> salary;

Storage::AddEmployee(Bookkeeper(firstName, middleName, lastName, personnelNumber, numberHourPerMonth, salary));

}

void Logic::SortByHours(string typeSort) {

Storage::SortByHours(typeSort);

}

void Logic::ReadFile() {

fstream file;

file.open("Storage.txt");

string firstName;

string middleName;

string lastName;

string personnelNumber;

int numberHourPerMonth;

int salary;

if (file.is\_open()) {

cout << "Файл открыт" << endl;

for (; !file.eof();)

{

file >> firstName >> middleName >> lastName >> personnelNumber >> numberHourPerMonth >> salary;

Storage::AddEmployee(Employee(firstName, middleName, lastName, personnelNumber, numberHourPerMonth, salary));

}

}

else {

cout << "Ошибка открытия файла" << endl;

}

Storage::DeleteEmployee(Storage::GetLength() - 1);

file.close();

}

void Logic::SaveFile() {

ofstream OutFile;

OutFile.open("Storage.txt");

if (OutFile.is\_open()) {

cout << "Saving was successful!" << "\n";

Sleep(2000);

for (int i = 0; i < Storage::GetLength(); i++) {

OutFile << Storage::GetEmployeeDataText(i) << endl;

}

}

else {

cout << "Ошибка открытия файла" << endl;

}

OutFile.close();

}

void Logic::EditEmployee(int id) {

Storage::EditEmployee(id);

}

Manager.cpp

#include "Manager.h"

void Manager::ShowInfo() {

cout << id << " " << "Manager" << " " << firstName << " " << lastName << " " << lastName << " " << numberHourPerMonth << endl;

}

Menu.cpp

#include "Menu.h"

void Menu::ShowMainMenu() {

Logic::ReadFile();

while (true) {

system("CLS");

int menuNumber;

cout << "Enter the number of menu to choose action" << endl << endl;

cout << "Main menu" << endl;

cout << "1. Save file" << endl;

cout << "2. Show data" << endl;

cout << "3. Add employee" << endl;

cout << "4. Delete employee" << endl;

cout << "5. Edit employee" << endl;

cout << "6. Sort by hours" << endl;

cout << "7. Calculate salary" << endl;

cout << "8. Exit" << endl;

cin >> menuNumber;

switch (menuNumber)

{

case 1: {

Logic::SaveFile();

break;

}

case 2: {

ShowDataMenu();

break;

}

case 3: {

ShowAddEmployeeMenu();

break;

}

case 4: {

ShowDeleteMenu();

break;

}

case 5: {

EditEmployeeMenu();

break;

}

case 6: {

ShowSortMenu();

break;

}

case 7: {

ShowCalculateSalary();

break;

}

case 8: {

exit(0);

}

default:

break;

}

}

}

void Menu::EditEmployeeMenu() {

while (true) {

system("CLS");

int employeeId;

Logic::ShowData();

cout << "Enter id to edit: ";

cin >> employeeId;

Logic::EditEmployee(employeeId);

break;

}

}

void Menu::ShowDeleteMenu() {

while (true)

{

system("CLS");

int employeeId;

Logic::ShowData();

cout << "Enter id to delete: ";

cin >> employeeId;

Logic::DeleteEmployee(employeeId);

break;

}

}

void Menu::ShowSortMenu() {

while (true)

{

system("CLS");

int menuNumber;

cout << "1. Ascending sort" << endl;

cout << "2. Descending sort" << endl;

cout << "3. Back" << endl;

cin >> menuNumber;

switch (menuNumber)

{

case 1: {

ShowAscendingSort();

break;

}

case 2: {

ShowDescendingSort();

break;

}

case 3: {

return;

}

default:

break;

}

}

}

void Menu::ShowAscendingSort() {

while (true)

{

system("CLS");

int menuNumber;

Logic::SortByHours("Ascending");

cout << "1. Back" << endl;

cin >> menuNumber;

switch (menuNumber)

{

case 1: {

return;

}

default:

break;

}

}

}

void Menu::ShowDescendingSort() {

while (true)

{

system("CLS");

int menuNumber;

Logic::SortByHours("Descending");

cout << "1. Back" << endl;

cin >> menuNumber;

switch (menuNumber)

{

case 1: {

return;

}

default:

break;

}

}

}

void Menu::ShowCalculateSalary() {

while (true)

{

system("CLS");

int menuNumber;

Logic::ShowSalary();

cout << "1. Back" << endl;

cin >> menuNumber;

switch (menuNumber)

{

case 1: {

return;

}

default:

break;

}

}

}

void Menu::ShowDataMenu() {

while (true)

{

system("CLS");

int menuNumber;

Logic::ShowData();

cout << "1. Back" << endl;

cin >> menuNumber;

switch (menuNumber)

{

case 1: {

return;

}

default:

break;

}

}

}

void Menu::ShowAddEmployeeMenu() {

while (true)

{

system("CLS");

int menuNumber;

cout << "1. Add manager" << endl;

cout << "2. Add engineer" << endl;

cout << "3. Add bookkeeper" << endl;

cout << "4. Back" << endl;

cin >> menuNumber;

switch (menuNumber)

{

case 1: {

Logic::AddManager();

break;

}

case 2: {

Logic::AddEngineer();

break;

}

case 3: {

Logic::AddBookKeeper();

break;

}

case 4: {

return;

}

default:

break;

}

}

}

Storage.cpp

#include "Storage.h"

vector<Employee> Storage::employees;

string Storage::GetEmployeeDataText(int i) {

return employees.at(i).GetDataText();

}

int Storage::GetLength() {

return employees.size();

}

void Storage::AddEmployee(Employee employee) {

employees.push\_back(employee);

}

void Storage::DeleteEmployee(int employeeId) {

auto iter = employees.begin();

employees.erase(iter + employeeId);

}

void Storage::ShowEmployees() {

if (employees.size() == 0) {

cout << "Empty data" << endl;

return;

}

cout << "ID" << setw(15) << "FIRSTNAME" << setw(15) << "MIDDLENAME" << setw(15) << "LASTNAME" << setw(15) << "P. NUMBER"

<< setw(15) << "HOURS" << setw(15) << "SALARY" << endl;

for (unsigned int i = 0; i < employees.size(); i++) {

employees.at(i).ShowInfo();

}

}

void Storage::ShowSalaryEmployeesWithTax() {

const int OVERTIME = 2;

if (employees.size() == 0) {

cout << "Empty data" << endl;

return;

}

cout << "ID" << setw(15) << "FIRSTNAME" << setw(15) << "MIDDLENAME" << setw(15) << "LASTNAME" << setw(15) << "P. NUMBER"

<< setw(15) << "HOURS" << setw(15) << "SALARY(with tax)" << endl;

for (unsigned int i = 0; i < employees.size(); i++) {

float salary;

if (employees.at(i).GetNumberHourPerMonth() > employees.at(i).GetFULL\_WORKING\_HOURS()) {

salary = employees.at(i).GetSalary() \* OVERTIME - employees.at(i).GetSalary() \* OVERTIME \* employees.at(i).GetINCOME\_TAX();

}

else {

salary = employees.at(i).GetSalary() - employees.at(i).GetSalary() \* employees.at(i).GetINCOME\_TAX();

}

cout << to\_string(employees.at(i).GetEmployeeId()) << setw(15) << employees.at(i).GetFirstName() << setw(15) << employees.at(i).GetMiddleName() << setw(15) <<

employees.at(i).GetLastName() << setw(15) << employees.at(i).GetPersonnelNumber() << setw(15) << to\_string(employees.at(i).GetNumberHourPerMonth()) <<

setw(15) << setprecision(5) << salary << endl;

}

}

void Storage::SortByHours(string typeSort) {

vector<Employee> tempEmployees = employees;

if (typeSort == "Ascending") {

for (int i = 0; i < tempEmployees.size()-1; i++)

{

for (int j = 0; j < tempEmployees.size()-i-1; j++)

{

if (tempEmployees.at(j).GetNumberHourPerMonth() > tempEmployees.at(j + 1).GetNumberHourPerMonth()) {

Employee temp = tempEmployees.at(j);

tempEmployees.at(j) = tempEmployees.at(j + 1);

tempEmployees.at(j + 1) = temp;

}

}

}

}

else {

for (int i = 0; i < tempEmployees.size() - 1; i++)

{

for (int j = 0; j < tempEmployees.size() - i - 1; j++)

{

if (tempEmployees.at(j).GetNumberHourPerMonth() < tempEmployees.at(j + 1).GetNumberHourPerMonth()) {

Employee temp = tempEmployees.at(j);

tempEmployees.at(j) = tempEmployees.at(j + 1);

tempEmployees.at(j + 1) = temp;

}

}

}

}

if (tempEmployees.size() == 0) {

cout << "Empty data" << endl;

return;

}

cout << "ID" << setw(15) << "FIRSTNAME" << setw(15) << "MIDDLENAME" << setw(15) << "LASTNAME" << setw(15) << "P. NUMBER"

<< setw(15) << "HOURS" << setw(15) << "SALARY" << endl;

for (unsigned int i = 0; i < tempEmployees.size(); i++) {

tempEmployees.at(i).ShowInfo();

}

tempEmployees.clear();

}

void Storage::EditEmployee(int id) {

bool check = false;

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

{

if (id == employees.at(i).GetId()) {

check = true;

break;

}

}

if (check == false) {

cout << "Id not found";

return;

}

else {

while(true) {

int menuNumber;

string text;

int number;

cout << "1. FIRSTNAME" << endl;

cout << "2. MIDDLENAME" << endl;

cout << "3. LASTNAME" << endl;

cout << "4. P. NUMBER" << endl;

cout << "5. HOURS" << endl;

cout << "6. SALARY" << endl;

cout << "7. Back" << endl;

cout << "Enter number of field that you whant change: ";

cin >> menuNumber;

switch (menuNumber)

{

case 1: {

cout << "FIRSTNAME: ";

cin >> text;

employees.at(id).SetFirstName(text);

break;

}

case 2: {

cout << "MIDDLENAME: ";

cin >> text;

employees.at(id).SetMiddleName(text);

break;

}

case 3: {

cout << "LASTNAME: ";

cin >> text;

employees.at(id).SetLastName(text);

break;

}

case 4: {

cout << "P. NUMBER: ";

cin >> text;

employees.at(id).SetPersonnelNumber(text);

break;

}

case 5: {

cout << "HOURS: ";

cin >> number;

employees.at(id).SetNumberHourPerMonth(number);

break;

}

case 6: {

cout << "P. NUMBER: ";

cin >> number;

employees.at(id).SetSalary(number);

break;

}

case 7: {

return;

}

default:

break;

}

}

}

}

Term paper.cpp

#include <iostream>

#include "Employee.h"

#include "Manager.h"

#include "Engineer.h"

#include "Bookkeeper.h"

#include "Menu.h"

#include "Storage.h"

using namespace std;

int main()

{

Menu::ShowMainMenu();

}