|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИТ)**

**Кафедра инструментального и прикладного программного обеспечения (ИиППО)**

**КУРСОВОЙ ПРОЕКТ (РАБОТА)**

по дисциплине: Объектно-ориентированное программирование

по профилю: Разработка программных продуктов и проектирование информационных систем

направления профессиональной подготовки: 09.03.04 “Программная инженерия” (академический бакалавриат)

Тема: «Программное обеспечение для управления численностью сотрудников для отдела кадров учебного заведения: класс сотрудник и класс структурных подразделений (отделов и др.) для отдела кадров организации»

Студент: Тимошкин Максим Сергеевич

Группа: ИКБО-13-18

Работа представлена к защите\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

(подпись и ф.и.о. студента)

Руководитель: Иванов Михаил Евгеньевич, ассистент кафедры ИППО

Работа допущена к защите\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

(подпись ф.и.о. руководителя)

Оценка по итогам защиты: \_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

М. МИРЭА. 2019г.

УДК 004.4

Тимошкин М.С.Программное обеспечение для управления численностью сотрудников для отдела кадров учебного заведения: класс сотрудник и класс структурных подразделений (отделов и др.) для отдела кадров организации **/ Курсовая работа** по дисциплине «Объектно-ориентированное программирование» профиля «Разработка программных продуктов и проектирование информационных систем» направления профессиональной подготовки академического бакалавриата 09.03.04. «Программная инженерия» (2ой семестр) / руководитель асс. М.Е. Иванов / кафедра ИиППО Института ИТ РТУ МИРЭА – с. 52, илл.12, ист. 3, (в т.ч. 1 на англ. яз.).

Целью работы является разработка программного обеспечения для управления численностью сотрудников для отдела кадров учебного заведения.

В рамках данной работы были разработаны логика программного обеспечения и структура базы данных. В рамках работы также были осуществлены анализ предметной области, проектирование и разработка приложения, а также проведено тестирование на стабильность работы приложения.

Timoshkin M.S. Software for headcount management for the personnel department of an educational institution: the employee class and the class of structural units (departments, etc.) for the personnel department of the organization / **Course work** for class "Object-oriented programming" of "Development program products and designing informational systems" educational profile of professional education of academic bachelor degree 09.03.04. "Program Engineering" (2nd semester) / Head Ass. M.E. Ivanov / Department of IIPPO Institute IT RTU MIREA - p. 52, ill.12, ist. 4, (including 1 in English. Lang.).

The aim of the work is the development of the software for headcount management for the personnel department of an educational institution.

As part of this work, software logic and database structure were developed. Within the framework of the work, the subject area analysis, design and development of the application were also carried out, as well as testing for software’s stability.

М. МИРЭА. Ин-т ИТ. Каф. ИиППО. 2019г. @ Тимошкин М.С.

**СОДЕРЖАНИЕ**

[СПИСОК ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ 6](#_Toc9169585)

[ВВЕДЕНИЕ 7](#_Toc9169586)

[Глава 1 Анализ программных средств 9](#_Toc9169587)

[1.1 «Exiland Assistant» Exiland Software 9](#_Toc9169588)

[1.1 Выводы к первой главе 10](#_Toc9169589)

[Глава 2 Методология разработки приложения 11](#_Toc9169590)

[2.1 Выбор языка программирования для разработки приложения 11](#_Toc9169591)

[2.2 Выбор инструментов для разработки функциональной системы приложения 11](#_Toc9169592)

[2.2.1 CLion IDE 11](#_Toc9169593)

[2.2.2 GitHub 11](#_Toc9169594)

[2.3 Разработка структуры приложения 11](#_Toc9169595)

[2.3.1 Базовая структура схемы работы программы 12](#_Toc9169596)

[2.3.2 Разработка базы данных для приложения 12](#_Toc9169597)

[2.4 Интерфейс приложения 13](#_Toc9169598)

[2.4.1 Главное меню 13](#_Toc9169599)

[2.4.2 Меню добавления сотрудника 14](#_Toc9169600)

[2.4.3 Меню изменения сотрудника 15](#_Toc9169601)

[2.4.4 Меню удаления сотрудника 16](#_Toc9169602)

[2.4.5 Меню вывода списка сотрудников 17](#_Toc9169603)

[2.4.6 Меню расчета средней з/п по всем видам сотрудников и отделам 17](#_Toc9169604)

[2.5 Выводы ко второй главе 18](#_Toc9169605)

[Глава 3 Реализация приложения 19](#_Toc9169606)

[3.1 UML-диаграммы классов 19](#_Toc9169607)

[3.2 Считывание списка сотрудников из файла 20](#_Toc9169608)

[3.3 Главное меню 21](#_Toc9169609)

[3.4 Меню добавления сотрудника 21](#_Toc9169610)

[3.5 Меню изменения сотрудника 23](#_Toc9169611)

[3.6 Меню удаления сотрудника 24](#_Toc9169612)

[3.7 Меню вывода полной информации о сотруднике 25](#_Toc9169613)

[3.8 Меню вывода средней з/п по всем видам сотрудников и институтам 25](#_Toc9169614)

[3.9 Опция выхода 28](#_Toc9169615)

[3.10 Выводы к третьей главе 28](#_Toc9169616)

[Глава 4 Тестирование приложения 29](#_Toc9169617)

[4.1 Выводы к четвертой главе 29](#_Toc9169618)

[ВЫВОД 30](#_Toc9169619)

[СПИСОК ИСТОЧНИКОВ 31](#_Toc9169620)

[ПРИЛОЖЕНИЕ А 32](#_Toc9169621)

СПИСОК ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ

«IDE» - «Integrated Development Environment», интегрированная среда разработки.

«ПО» - программное обеспечение.

«ООП» - объектно-ориентированное программирование.

«IT» – Information Technologies, информационные технологии.

«з/п» – заработная плата.

«БД» – база данных.

«ИИТ» - институт информационных технологий.

«ФТИ» - физико-технологический институт.

«Киб» - институт кибернетики.

ВВЕДЕНИЕ

В современном мире различные организации насчитывают огромное количество работников на своих предприятиях. Держать информацию о каждом из них в голове достаточно трудно, можно с легкостью забыть какие-либо важные аспекты кого-нибудь из сотрудников.

Раньше человек использовал для хранения информации о сотрудниках бумажные ресурсы. Сейчас же, в век развития информационных технологий, это представляется крайне трудоемким и неудобным средством хранения информации. Вот ряд причин, по которым фиксирование информации о сотрудниках на бумажных ресурсах не является практичным:

1. Хранение на бумажных носителях информации об огромном количестве сотрудников не экономно по отношению к использованию бумаги.
2. Навигация по существующим сотрудникам и поиск нужного занимает много времени.
3. Расчет каких-либо характеристик, например средней заработной платы сотрудников определенного отдела, занимает много времени, так как требует поиска сотрудников по конкретному критерию.
4. Редактирование сотрудников представляет собой сложную комплексную задачу по поиску сотрудника и редактированию информации на бумажном носителе, что выглядит не аккуратно.
5. Добавление новых сотрудников требует дополнительных бумажных ресурсов, что не экономно.

После развития компьютерных и информационных технологий были созданы электронные аналоги бумажных картотек, которые во много раз практичнее и удобнее.

Именно поэтому разработку программного обеспечения для управления численностью сотрудников я и выбрал в качестве темы для своей курсовой работы.

При разработке курсовой работы планирует приобрести такие компетенции ОПК-1, ПК-14 и ПК-20.

Глава 1 Анализ программных средств

Картотека сотрудников – упорядоченное собрание данных о сотрудниках того или иного предприятия. Идея разработки электронной картотеки не нова, поэтому существует немалое количество различных программ для ведения электронной картотеки.

Можно проанализировать некоторые из них для выделения основного необходимого функционала.

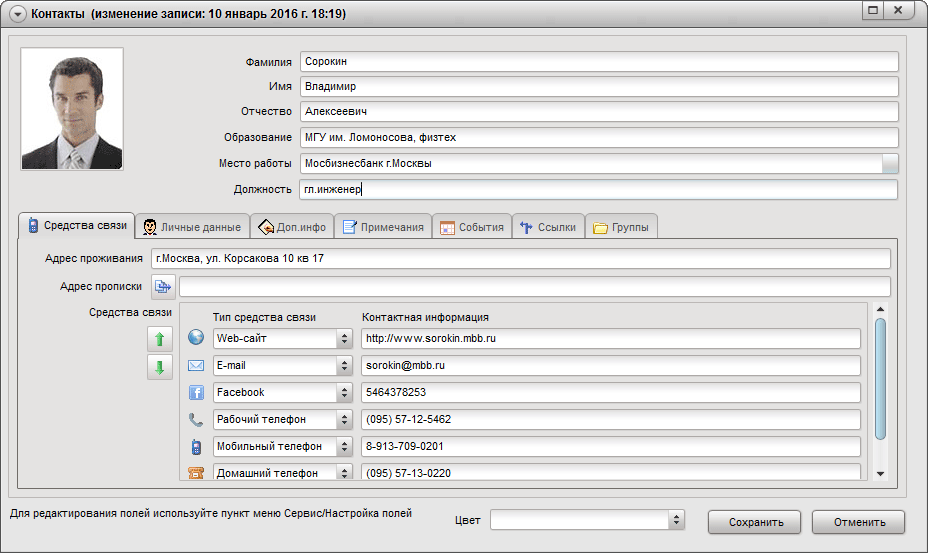
# 1.1 «Exiland Assistant» Exiland Software

Exiland Assistant – универсальный ежедневный органайзер, который пригоден для решения огромного спектра задач.

Программа позволяет вести:

* подробную базу по людям: клиентам, сотрудникам, друзьям;
* информацию по организациям;
* заметки;
* ссылки;
* задачи и события.

Электронная картотека позволяет хранить множество различной информации. (Рисунок 1.1)



**Рисунок 1.1** – Картотека человека в Exiland Assistant

Органайзер позволяет хранить следующие данные о человеке:

* Ф.И.О;
* фотографию;
* место работы, должность;
* адреса проживания и прописки.

# Выводы к первой главе

На основании рассмотренного выше программного обеспечения, для реализации электронной картотеки для управления численностью сотрудников для отдела кадров учебного заведения ставятся задачи:

1. Спроектировать функции, которыми должна обладать программа.
2. Разработать интуитивно-понятное, многофункциональное ПО, обладающее простым интерфейсом.
3. Произвести тестирование разработанного продукта.

Глава 2 Методология разработки приложения

# 2.1 Выбор языка программирования для разработки приложения

В качестве основного языка для разработки ПО был выбран язык C++, из семейства Объектно-ориентированных языков, так как обучение основам ООП в этом семестре ведется на примере именно этого языка.

# 2.2 Выбор инструментов для разработки функциональной системы приложения

# 2.2.1 CLion IDE

CLion — умная IDE, предназначенная для разработки на С и С++ на платформах Linux, Windows и macOS. Включает в себя поддержку системы сборки кросс-платформенных проектов CMake, Gradle C++ и compilation database, возможности анализа кода на лету, разнообразные рефакторинги и унифицированный интерфейс для работы с самыми популярными системами контроля версий.

# 2.2.2 GitHub

GitHub – бесплатный интернет-ресурс для разработки и хостинга IT-проектов, с встроенной системой контроля приложений.

GitHub был выбран как сервис по поддержанию контроля версий приложения.

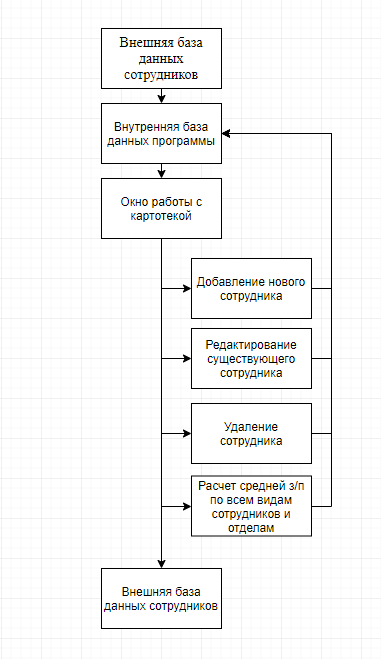
# 2.3 Разработка структуры приложения

Программа содержит следующие функции:

* добавление нового сотрудника;
* редактирование информации уже существующего сотрудника;
* удаление сотрудника;
* вывод списка сотрудников, получение полной информации выбранного сотрудника;
* расчет средней з/п по различным видам сотрудников;
* расчет средней з/п по различным институтам образовательного учреждения.

# 2.3.1 Базовая структура схемы работы программы

Базовая структура схемы работы программы представлена на рисунке 2.1.



**Рисунок 2.1** – Базовая структура схемы

работы программы

Внешняя база данных сотрудников, которая располагается на внешнем носителе информации, соединяется с внутренней базой данных программы, которая создается в момент запуска программы и удаляется из памяти в момент закрытия программы. Происходит копирование БД с внешнего носителя во внутреннюю память программы. После успешных преобразований в БД, она копируется в БД на внешнем носителе.

# 2.3.2 Разработка базы данных для приложения

В БД содержатся следующие сведения:

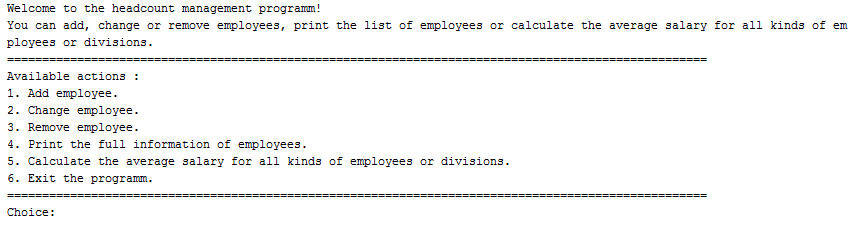
* имя сотрудника;
* должность сотрудника;
* институт, к которому относится сотрудник;
* з/п сотрудника в соответствии с занимаемой им должностью.

# 2.4 Интерфейс приложения

# 2.4.1 Главное меню

На рисунке 2.2 представлено главное меню программы. Используя цифры от 1 до 6 можно выбрать одну из 6 главных опций программы:

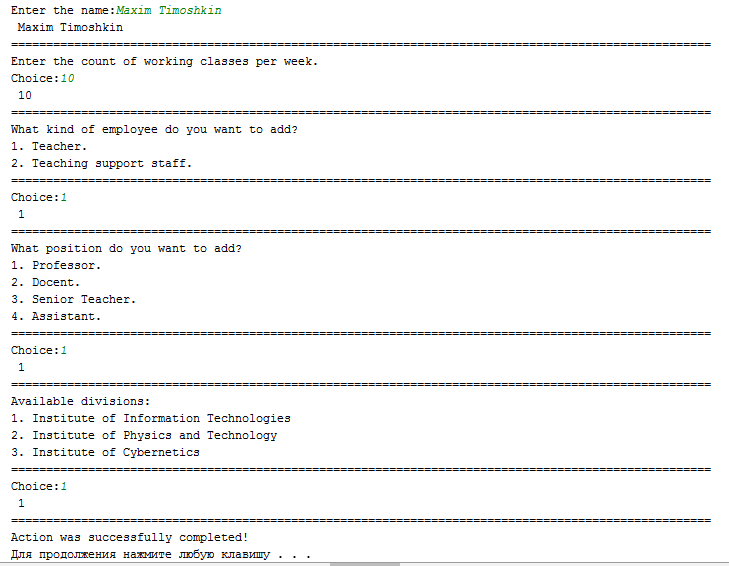
1. Add employee – добавить сотрудника. Данная опция позволяет добавить сотрудника в БД, выбрав для него имя, должность, институт и з/п.
2. Change employee – изменить сотрудника. Данная опция позволяет изменить такую информацию о сотруднике, как имя, должность, институт или з/п.
3. Remove employee – удалить сотрудника. Данная опция позволяет удалить сведения о сотруднике из БД.
4. Print the full information of employees – напечатать полную информацию о сотрудниках. Данная опция позволяет посмотреть основную информацию какого-либо из сотрудников, такую как имя, должность, з/п и институт.
5. Calculate the average salary for all kinds of employees or divisions – посчитать среднюю з/п по всем видам сотрудников или отделам. Данная опция позволяет посчитать и отобразить среднюю з/п по различным должностям, по всем сотрудникам и по различным институтам.
6. Exit the program – выйти из программы. Данная опция копирует данные из внутренней БД во внешнюю, очищает память внутренней БД и выходит из программы.



**Рисунок 2.2** – Главное меню программы

# 2.4.2 Меню добавления сотрудника

На рисунке 2.3 представлено меню добавления сотрудника.



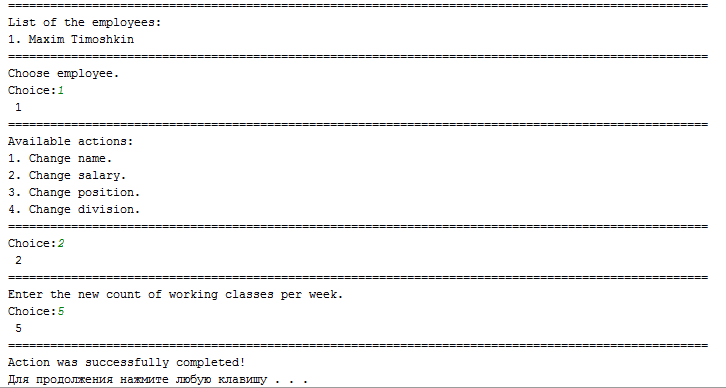
**Рисунок 2.3** – Меню добавления сотрудника

При выборе пункта «Add employee» пользователю предоставляется возможность добавить сотрудника, введя:

1. Имя сотрудника.
2. Количество рабочих часов в неделю (необходимый критерий для подсчета з/п).
3. Должность сотрудника (определяет размер з/п):
   1. преподаватель;
      * профессор;
      * доцент;
      * старший преподаватель;
      * ассистент.
   2. учебно-вспомогательный персонал.
      * инженер;
      * лаборант.
4. Институт, к которому относится добавленный сотрудник:
   1. ИИТ;
   2. ФТИ;
   3. Киб.

# 2.4.3 Меню изменения сотрудника

На рисунке 2.4 представлено меню изменения сотрудника.



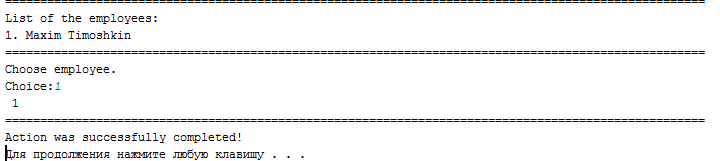
**Рисунок 2.4** – Меню изменения сотрудника

При выборе пункта «Change employee» пользователю предоставляется возможность изменить всю имеющуюся информацию о сотруднике:

1. Имя сотрудника.
2. Количество рабочих часов в неделю (необходимый критерий для подсчета з/п).
3. Должность сотрудника (определяет размер з/п):
   1. преподаватель;
      * профессор;
      * доцент;
      * старший преподаватель;
      * ассистент.
   2. учебно-вспомогательный персонал.
      * инженер;
      * лаборант.
4. Институт, к которому относится добавленный сотрудник:
   1. ИИТ;
   2. ФТИ;
   3. Киб.

# 2.4.4 Меню удаления сотрудника

На рисунке 2.5 представлено меню удаления сотрудника.

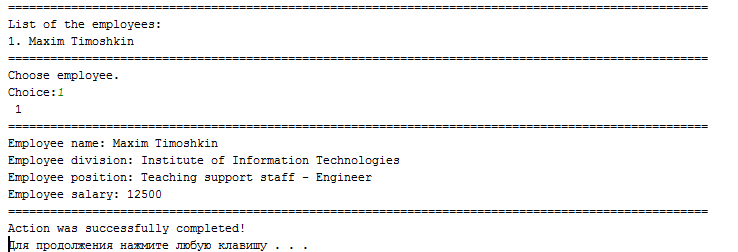


**Рисунок 2.5** – Меню удаления сотрудника

При выборе пункта «Remove employee» пользователю предоставляется возможность удалить сотрудника из списка существующих.

# 2.4.5 Меню вывода списка сотрудников

На рисунке 2.6 представлено меню вывода списка сотрудников

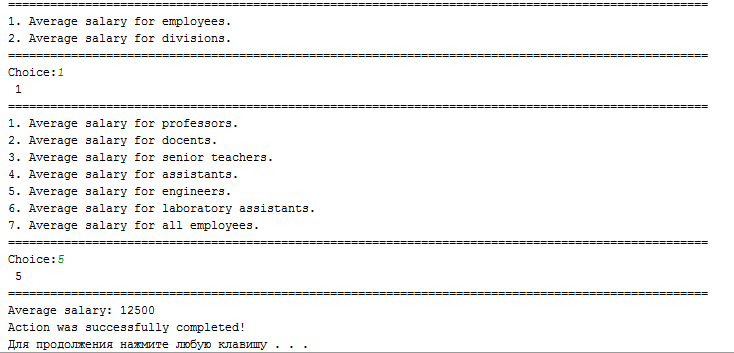


**Рисунок 2.6** – Меню вывода списка сотрудников

При выборе пункта «Print the full information of employees» пользователю предлагается выбрать из списка сотрудников интересующего для вывода полной информации о нем.

# 2.4.6 Меню расчета средней з/п по всем видам сотрудников и отделам

На рисунке 2.7 представлено меню расчета средней з/п по всем видам сотрудников и отделам.



**Рисунок 2.7** – Меню расчета средней з/п по всем видам сотрудников и отделам

Выбрав пункт «Calculate the average salary for all kinds of employees or divisions» пользователю предоставляется возможность рассчитать среднюю з/п по:

1. Сотрудникам.
   * + по всем профессорам;
     + по всем доцентам;
     + по всем старшим преподавателям;
     + по всем ассистентам;
     + по всем инженерам;
     + по всем лаборантам;
     + по всем сотрудникам.
2. Институтам.
   * + по всем сотрудникам ИИТ;
     + по всем сотрудникам ФТИ;
     + по всем сотрудникам Киб.

# 2.5 Выводы ко второй главе

Во второй главе:

Были рассмотрены программные средства, которые потребуются для разработки ПО.

Были разработаны модели внешней и внутренней БД.

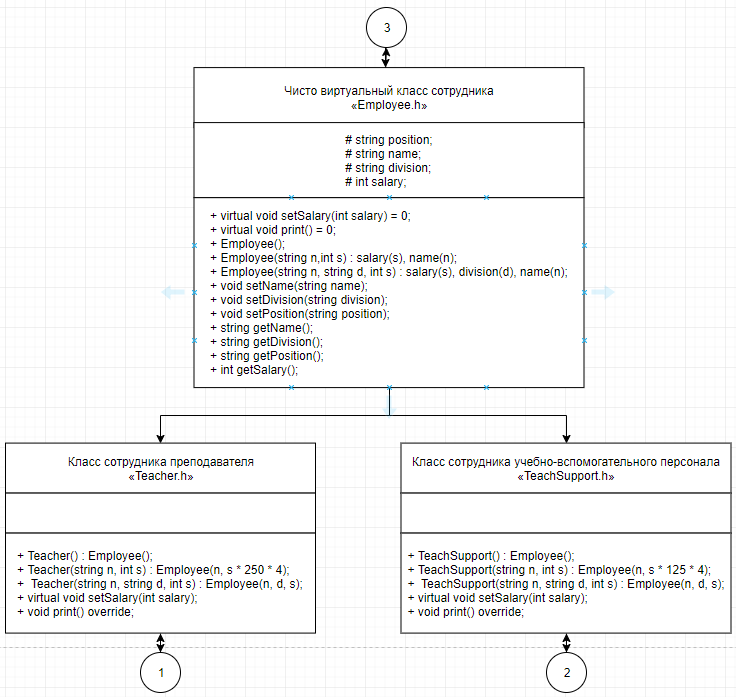
Был определен перечень базовых функций программы

Была разработана базовая схема работы программы.

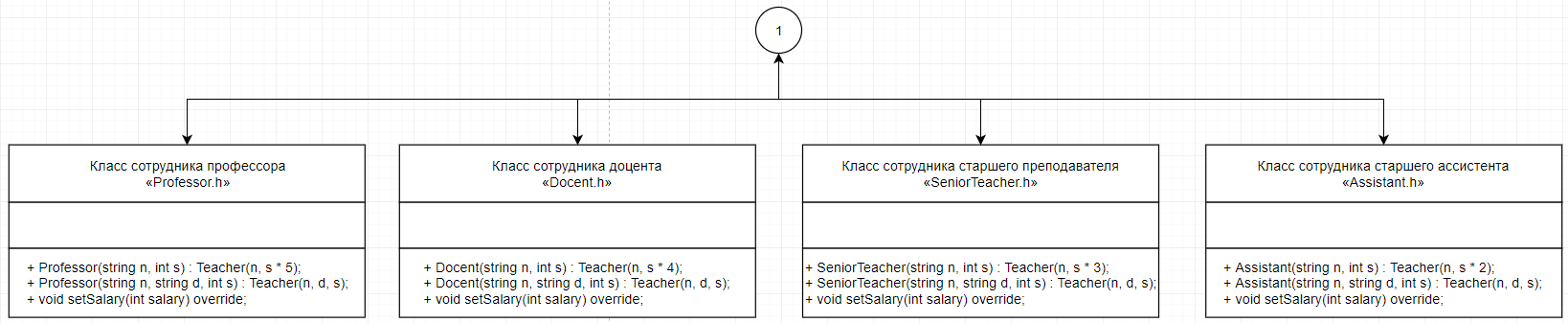
Было проведено ознакомление с интерфейсом программы.

Глава 3 Реализация приложения

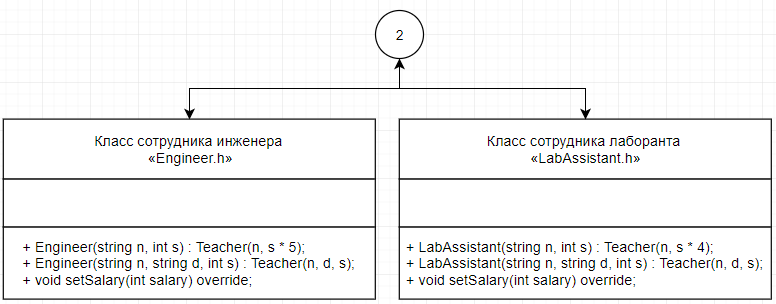
# 3.1 UML-диаграммы классов



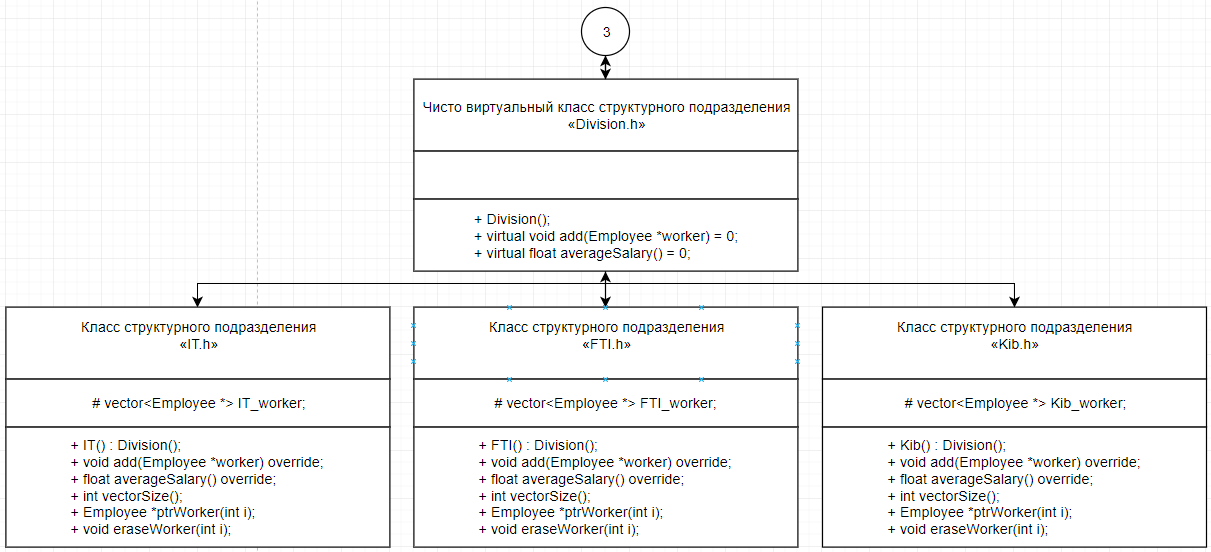
**Рисунок 3.1** – Диаграмма иерархии классов сотрудников



**Рисунок 3.2** – Диаграмма иерархии классов сотрудников



**Рисунок 3.3** – Диаграмма иерархии классов сотрудников



**Рисунок 3.4** – Диаграмма иерархии классов структурных подразделений

# Считывание списка сотрудников из файла

При запуске программы, происходит выполнение функции readTxt, которая считывает данные сотрудников из файла Employee.txt и создает его, если он не существует. При ошибке при чтении данных программа завершит свою работу с ошибкой.

**if** (readTxt(emp, InfT, Ft, Ki)) {  
 system(**"pause"**);  
 **return** 0;  
}

# Главное меню

После чтения данных выводится главное меню, с помощью которого пользователь может воспользоваться основным функционалом программы. Пользователю предлагается выбрать опцию. Благодаря функции correctInput происходит проверка на ввод только чисел.

cout << **"Welcome to the headcount management programm!"** << endl  
 << **"You can add, change or remove employees, print the list of employees or calculate the average salary for all kinds of employees or divisions."** << endl;  
  
printSeparator();  
  
cout << **"Available actions :"** << endl;  
cout << **"1. Add employee."** << endl;  
cout << **"2. Change employee."** << endl;  
cout << **"3. Remove employee."** << endl;  
cout << **"4. Print the full information of employees."** << endl;  
cout << **"5. Calculate the average salary for all kinds of employees or divisions."** << endl;  
cout << **"6. Exit the programm."** << endl;  
printSeparator();  
**do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 6)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
} **while** ((buffSwtch < 1) || (buffSwtch > 6));

# Меню добавления сотрудника

В меню добавления сотрудника пользователь выбирает имя, размер з/п, должность и институт, к которому данный сотрудник относится. Сотрудник добавляется в вектор указателей класса Employee emp и в зависимости от выбранного института в один из векторов указателей класса Employee InfT, Ft или Ki.

**case** 1: {  
 cout << **"What position do you want to add?"** << endl;  
 cout << **"1. Professor."** << endl;  
 cout << **"2. Docent."** << endl;  
 cout << **"3. Senior Teacher."** << endl;  
 cout << **"4. Assistant."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 4)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 4));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 emp.push\_back(**new** Professor(buff, buffInt));  
 **break**;  
 }  
 **case** 2: {  
 emp.push\_back(**new** Docent(buff, buffInt));  
 **break**;  
 }  
 **case** 3: {  
 emp.push\_back(**new** SeniorTeacher(buff, buffInt));  
 **break**;  
 }  
 **case** 4: {  
 emp.push\_back(**new** Assistant(buff, buffInt));  
 **break**;  
 }  
 }  
 **break**;  
 }  
 **case** 2: {  
 cout << **"What position do you want to add?"** << endl;  
 cout << **"1. Engineer."** << endl;  
 cout << **"2. Laboratory Assistant."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 2)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 2));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 emp.push\_back(**new** Engineer(buff, buffInt));  
 **break**;  
 }  
 **case** 2: {  
 emp.push\_back(**new** LabAssistant(buff, buffInt));  
 **break**;  
 }  
 }  
 **break**;  
 }  
  
 }  
 cout << **"Available divisions: "** << endl;  
 cout << **"1. Institute of Information Technologies"** << endl;  
 cout << **"2. Institute of Physics and Technology"** << endl;  
 cout << **"3. Institute of Cybernetics"** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 3)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 3));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 emp[emp.size() - 1]->setDivision(**"Institute of Information Technologies"**);  
 InfT.add(\*(emp.end() - 1));  
 **break**;  
 }  
 **case** 2: {  
 emp[emp.size() - 1]->setDivision(**"Institute of Physics and Technology"**);  
 Ft.add(\*(emp.end() - 1));  
 **break**;  
 }  
 **case** 3: {  
 emp[emp.size() - 1]->setDivision(**"Institute of Cybernetics"**);  
 Ki.add(\*(emp.end() - 1));  
 **break**;  
 }  
 }  
 completeOperation();  
 **break**;  
}

# 3.5 Меню изменения сотрудника

В меню изменения сотрудника пользователь может изменить любой из доступных параметров: имя, размер з/п, институт или должность. С помощью функции changePosition происходит изменение должности сотрудника на выбранную. При этом номер сотрудника в общем списке остается неизменным. Также при изменении должности учитывается различный оклад должностей, происходит изменение з/п в соответствии с должностью. С помощью функции isExist происходит проверка наличия сотрудников в списке. При их отсутствии программа возвращает пользователя в главное меню.

case 2: {  
 **if** (!(isExist(emp))) {  
 cout << **"There aren't any employees!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 printEmp(emp);  
 cout << **"Choose employee."** << endl;  
 **do** {  
 buffInt = correctInput();  
 **if** ((buffInt < 1) || (buffInt > emp.size())) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffInt < 1) || (buffInt > emp.size()));  
 cout << **"Available actions: "** << endl;  
 cout << **"1. Change name."** << endl;  
 cout << **"2. Change salary."** << endl;  
 cout << **"3. Change position."** << endl;  
 cout << **"4. Change division."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 4)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 4));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 cin.ignore();  
 cout << **"Enter the new name: "**;  
 getline(cin, buff);  
 emp[buffInt - 1]->setName(buff);  
 completeOperation();  
 **break**;  
 }  
 **case** 2: {  
 cout << **"Enter the new count of working classes per week."** << endl;  
 buffSal = correctInput();  
 emp[buffInt - 1]->setSalary(buffSal);  
 completeOperation();  
 **break**;  
 }  
 **case** 3: {  
 changePosition(emp, InfT, Ft, Ki, buffInt);  
 completeOperation();  
 **break**;  
 }  
  
 **case** 4: {  
 changeDivision(emp, InfT, Ft, Ki, buffInt);  
 completeOperation();  
 **break**;  
 }  
 }  
 **break**;

}

# 3.6 Меню удаления сотрудника

В меню удаления сотрудника пользователю предлагается выбрать сотрудника для удаления. При выборе данной опции список также проверяется на наличие сотрудников с помощью функции isExist.

**case** 3: {  
 **if** (!(isExist(emp))) {  
 cout << **"There aren't any employees!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 printEmp(emp);  
 cout << **"Choose employee."** << endl;  
 **do** {  
 buffInt = correctInput();  
 **if** ((buffInt < 1) || (buffInt > emp.size())) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffInt < 1) || (buffInt > emp.size()));  
 deleteEmp(emp, InfT, Ft, Ki, buffInt);  
 completeOperation();  
 **break**;  
}

# 3.7 Меню вывода полной информации о сотруднике

При выборе данной опции пользователю предлагается выбрать из списка существующих сотрудников интересующего для вывода полной информации о нем.

**case** 4: {  
 **if** (!(isExist(emp))) {  
 cout << **"There aren't any employees!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 printEmp(emp);  
 cout << **"Choose employee."** << endl;  
 **do** {  
 buffInt = correctInput();  
 **if** ((buffInt < 1) || (buffInt > emp.size())) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffInt < 1) || (buffInt > emp.size()));  
 emp[buffInt - 1]->print();  
 printSeparator();  
 completeOperation();  
 **break**;  
}

# 3.8 Меню вывода средней з/п по всем видам сотрудников и институтам

В данном меню пользователю предлагается выбрать по какой из доступных должностей рассчитать среднюю з/п. Также можно рассчитать среднюю з/п по всем сотрудникам или по сотрудникам из отдельных институтов. При отсутствии сотрудников в той группе, по которой производится расчет средней з/п, пользователя возвращают в главное меню.

**case** 5: {  
 **if** (!(isExist(emp))) {  
 cout << **"There aren't any employees!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"1. Average salary for employees."** << endl;  
 cout << **"2. Average salary for divisions."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 2)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 2));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 cout << **"1. Average salary for professors."** << endl;  
 cout << **"2. Average salary for docents."** << endl;  
 cout << **"3. Average salary for senior teachers."** << endl;  
 cout << **"4. Average salary for assistants."** << endl;  
 cout << **"5. Average salary for engineers."** << endl;  
 cout << **"6. Average salary for laboratory assistants."** << endl;  
 cout << **"7. Average salary for all employees."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 7)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 7));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 buff = **"Professor"**;  
 **if** (!(isExistPos(emp, buff))) {  
 cout << **"There aren't any "** << buff << **"s in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary: "** << averageSalary(emp, buff) << endl;  
 completeOperation();  
 **break**;  
 }  
 **case** 2: {  
 buff = **"Docent"**;  
 **if** (!(isExistPos(emp, buff))) {  
 cout << **"There aren't any "** << buff << **"s in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary: "** << averageSalary(emp, buff) << endl;  
 completeOperation();  
 **break**;  
 }  
 **case** 3: {  
 buff = **"Senior Teacher"**;  
 **if** (!(isExistPos(emp, buff))) {  
 cout << **"There aren't any "** << buff << **"s in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary: "** << averageSalary(emp, buff) << endl;  
 completeOperation();  
 **break**;  
 }  
 **case** 4: {  
 buff = **"Assistant"**;  
 **if** (!(isExistPos(emp, buff))) {  
 cout << **"There aren't any "** << buff << **"s in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary: "** << averageSalary(emp, buff) << endl;  
 completeOperation();  
 **break**;  
 }  
 **case** 5: {  
 buff = **"Engineer"**;  
 **if** (!(isExistPos(emp, buff))) {  
 cout << **"There aren't any "** << buff << **"s in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary: "** << averageSalary(emp, buff) << endl;  
 completeOperation();  
 **break**;  
 }  
 **case** 6: {  
 buff = **"Laboratory Assistant"**;  
 **if** (!(isExistPos(emp, buff))) {  
 cout << **"There aren't any "** << buff << **"s in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary: "** << averageSalary(emp, buff) << endl;  
 completeOperation();  
 **break**;  
 }  
 **case** 7: {  
 buffSal = 0;  
 **for** (**int** i = 0; i < emp.size(); i++)  
 buffSal += emp[i]->getSalary();  
 cout << **"Average salary: "** << **float**(buffSal) / emp.size() << endl;  
 completeOperation();  
 **break**;  
 }  
 }  
  
 **break**;  
 }  
 **case** 2: {  
 cout << **"1. Average salary for Institute of Inforamation Technologies."** << endl;  
 cout << **"2. Average salary for Institute of Physics and Technology."** << endl;  
 cout << **"3. Average salary for Institute of Cybernetics."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 3)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 3));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 **if** (InfT.vectorSize() == 0) {  
 cout << **"There aren't any IT workers in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary : "** << InfT.averageSalary() << endl;  
 completeOperation();  
 **break**;  
 }  
 **case** 2: {  
 **if** (Ft.vectorSize() == 0) {  
 cout << **"There aren't any FTI workers in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary : "** << Ft.averageSalary() << endl;  
 completeOperation();  
 **break**;  
 }  
 **case** 3: {  
 **if** (Ki.vectorSize() == 0) {  
 cout << **"There aren't any Kib workers in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary : "** << Ki.averageSalary() << endl;  
 completeOperation();  
 **break**;  
 }  
 }  
 **break**;  
 }  
 }  
 **break**;  
}

# 3.9 Опция выхода

При выходе из программы происходит запись в Employee.txt обновленного списка сотрудников, а также очистка указателей для избежания утечки памяти.

**case** 6: {  
 writeTxt(emp);  
 deletePtr(emp);  
  
 **return** 0;  
}

# 3.10 Выводы к третьей главе

В третьей главе были продемонстрированы uml-диаграммы классов и разработаны базовые функции приложения.

Глава 4 Тестирование приложения

В ходе разработки ПО были проведены различные тесты по которым находились и исправлялись некоторые ошибки приложения.

Тестирования проводились во время всей разработки приложения, на различных компьютерах на операционной системе Windows 10.

Тестовые запуски показали, что приложение:

* работоспособно;
* устойчиво к некорректному вводу, к ошибкам при чтении данных из файла;
* стабильно, перенос на разные компьютеры не влияет на скорость и эффективность работы приложения;
* интуитивно понятно, благодаря простому интерфейсу программы;
* функционально, приложение выполнено полностью в соответствии с техническим заданием.

# 4.1 Выводы к четвертой главе

В ходе выполнений курсовой работы было разработано полностью соответствующее техническому заданию эффективное, многофункциональное, интуитивно понятное приложение для управления численностью сотрудников для отдела кадров учебного заведения.

ВЫВОД

По итогам разработки были выполнены следующие задачи:

* была выбрана программная среда, а также спроектирована и разработана структура приложения и баз данных;
* была спроектирована логика и интерфейс приложения;
* было разработано ПО для управления численностью сотрудников для отдела кадров учебного заведения;
* для приложения был разработан интуитивно понятный интерфейс, не вызывающий трудностей в работе;
* все функции приложения, указанные в техническом задании, были реализованы и работают без ошибок;
* были произведены тесты, для оценки стабильности и работоспособности приложения;
* поставленная цель курсовой работы была выполнена.

В результате проведенной работы были изучены и приобретены следующие компетенции:

* ОПК-1 – владение основными компетенциями, принципами, теориями и фактами, связанными с информатикой;
* ПК-14 – готовностью обосновывать принимаемые проектные решения, осуществлять постановку и выполнение экспериментов по проверке их корректности и эффективности;
* ПК-20 – способностью оценивать временную и емкостную сложность программного обеспечения.

СПИСОК ИСТОЧНИКОВ

1. Страуструп Б. “Язык программирования С++, краткий курс” [Книжное издание 2017 год]
2. Лекционный курс по ООП за 2 семестр 1 курса.
3. Язык программирования C++ - https://en.cppreference.com/w/ [Электронный ресурс]

Обращение к указанным источникам происходило вплоть до финальной стадии разработки приложения.

ПРИЛОЖЕНИЕ А

**Исходный код программного обеспечения.**

# Employee.h

#pragma once  
  
#include **<string>**#include**<iostream>  
  
using namespace** std;  
  
**class** Employee {  
**protected**:  
 string position;  
 string name;  
 string division;  
 **int** salary;  
  
**public**:  
 **virtual void** setSalary(**int** salary) = 0;  
  
 **virtual void** print() = 0;  
  
 Employee() {  
 position = **"default"**;  
 name = **"default"**;  
 division = **"default"**;  
 salary = -1;  
 };  
  
 Employee(string n, **int** s) : salary(s), name(n) {};  
  
 Employee(string n, string d, **int** s) : salary(s), division(d), name(n) {};  
  
  
 **void** setName(string name) {  
 **this**->name = name;  
 }  
  
 **void** setDivision(string division) {  
 **this**->division = division;  
 }  
  
 **void** setPosition(string position) {  
 **this**->position = position;  
 }  
  
 string getName() {  
 **return** name;  
 }  
  
 string getDivision() {  
 **return** division;  
 }  
  
 string getPosition() {  
 **return** position;  
 }  
  
 **int** getSalary() {  
 **return** salary;  
 }  
};

# Teacher.h

#pragma once  
  
#include **"Employee.h"  
  
class** Teacher : **public** Employee {  
**public**:  
 Teacher() : Employee() {};  
  
 Teacher(string n, **int** s) : Employee(n, s \* 250 \* 4) {};  
  
 Teacher(string n, string d, **int** s) : Employee(n, d, s) {};  
  
  
 **virtual void** setSalary(**int** salary) {  
 **this**->salary = salary \* 250 \* 4;  
 }  
  
  
 **void** print() override {  
 cout << **"Employee name: "** << name << endl;  
 cout << **"Employee division: "** << division << endl;  
 cout << **"Employee position: Teacher - "** << position << endl;  
 cout << **"Employee salary: "** << salary << endl;  
 }  
};

# TeachSupport.h

#pragma once  
  
#include **"Employee.h"  
  
class** TeachSupport : **public** Employee {  
**public**:  
 TeachSupport() : Employee() {};  
  
 TeachSupport(string n, **int** s) : Employee(n, s \* 125 \* 4) {};  
  
 TeachSupport(string n, string d, **int** s) : Employee(n, d, s) {};  
  
 **virtual void** setSalary(**int** salary) {  
 **this**->salary = salary \* 125 \* 4;  
 }  
  
 **void** print() override {  
 cout << **"Employee name: "** << name << endl;  
 cout << **"Employee division: "** << division << endl;  
 cout << **"Employee position: Teaching support staff - "** << position << endl;  
 cout << **"Employee salary: "** << salary << endl;  
 }  
};

# Professor.h

#pragma once  
  
#include **"Teacher.h"  
  
class** Professor : **public** Teacher {  
  
**public**:  
 Professor(string n, **int** s) : Teacher(n, s \* 5) {  
 position = **"Professor"**;  
  
 };  
  
 Professor(string n, string d, **int** s) : Teacher(n, d, s) {  
 position = **"Professor"**;  
  
 };  
  
 **void** setSalary(**int** salary) override {  
 Teacher::setSalary(salary);  
  
 **this**->salary \*= 5;  
 }  
};

# Docent.h

#pragma once  
  
#include **"Teacher.h"  
  
class** Docent : **public** Teacher {  
**public**:  
 Docent(string n, **int** s) : Teacher(n, s \* 4) {  
 position = **"Docent"**;  
 };  
  
 Docent(string n, string d, **int** s) : Teacher(n, d, s) {  
 position = **"Docent"**;  
 };  
  
  
 **void** setSalary(**int** salary) override {  
 Teacher::setSalary(salary);  
  
 **this**->salary \*= 4;  
 }  
};

# Senior Teacher.h

#pragma once  
  
#include **"Teacher.h"  
  
class** SeniorTeacher : **public** Teacher {  
**public**:  
 SeniorTeacher(string n, **int** s) : Teacher(n, s \* 3) {  
 position = **"Senior Teacher"**;  
 };  
  
 SeniorTeacher(string n, string d, **int** s) : Teacher(n, d, s) {  
 position = **"Senior Teacher"**;  
 };  
  
  
 **void** setSalary(**int** salary) override {  
 Teacher::setSalary(salary);  
  
 **this**->salary \*= 3;  
 }  
};

# Assistant.h

#pragma once  
  
#include **"Teacher.h"  
  
class** Assistant : **public** Teacher {  
**public**:  
 Assistant(string n, **int** s) : Teacher(n, s \* 2) {  
 position = **"Assistant"**;  
 };  
  
 Assistant(string n, string d, **int** s) : Teacher(n, d, s) {  
 position = **"Assistant"**;  
 };  
  
 **void** setSalary(**int** salary) override {  
 Teacher::setSalary(salary);  
  
 **this**->salary \*= 2;  
 }  
};

# Engineer.h

#pragma once  
  
#include **"TeachSupport.h"  
  
class** Engineer : **public** TeachSupport {  
**public**:  
 Engineer(string n, **int** s) : TeachSupport(n, s \* 5) {  
 position = **"Engineer"**;  
 };  
  
 Engineer(string n, string d, **int** s) : TeachSupport(n, d, s) {  
 position = **"Engineer"**;  
 };  
  
 **void** setSalary(**int** salary) override {  
 TeachSupport::setSalary(salary);  
  
 **this**->salary \*= 5;  
 }  
};

# LabAssistant.h

#pragma once  
  
#include **"TeachSupport.h"  
  
class** LabAssistant : **public** TeachSupport {  
**public**:  
 LabAssistant(string n, **int** s) : TeachSupport(n, s \* 4) {  
 position = **"Laboratory Assistant"**;  
 };  
  
 LabAssistant(string n, string d, **int** s) : TeachSupport(n, d, s) {  
 position = **"Laboratory Assistant"**;  
 };  
  
 **void** setSalary(**int** salary) override {  
 TeachSupport::setSalary(salary);  
  
 **this**->salary \*= 4;  
 }  
};

# Division.h

#pragma once  
  
#include **<vector>**#include **"Assistant.h"**#include **"Docent.h"**#include **"Engineer.h"**#include **"LabAssistant.h"**#include **"Professor.h"**#include **"SeniorTeacher.h"  
  
using namespace** std;  
  
**class** Division {  
**public**:  
 Division() {};  
  
 **virtual void** add(Employee \*worker) = 0;  
  
 **virtual float** averageSalary() = 0;  
  
};

# IT.h

#pragma once  
  
#include **"Division.h"  
  
class** IT : **public** Division {  
**protected**:  
 vector<Employee \*> IT\_worker;  
**public**:  
 IT() : Division() {};  
  
 **void** add(Employee \*worker) override {  
 IT\_worker.push\_back(worker);  
 }  
  
 **float** averageSalary() override {  
 **float** n;  
 **for** (**int** i = 0; i < IT\_worker.size(); i++) {  
 n += IT\_worker[i]->getSalary();  
 }  
 n = n / (IT\_worker.size());  
 **return** n;  
 }  
  
 **int** vectorSize() {  
 **return** IT\_worker.size();  
 }  
  
 Employee \*ptrWorker(**int** i) {  
 **return** IT\_worker[i];  
 }  
  
 **void** eraseWorker(**int** i) {  
 IT\_worker.erase(IT\_worker.begin() + i);  
 }};

# FTI.h

#pragma once  
  
#include **"Division.h"  
  
class** FTI : **public** Division {  
**protected**:  
 vector<Employee \*> FTI\_worker;  
**public**:  
 FTI() : Division() {};  
  
 **void** add(Employee \*worker) override {  
 FTI\_worker.push\_back(worker);  
 }  
  
 **float** averageSalary() override {  
 **float** n;  
 **for** (**int** i = 0; i < FTI\_worker.size(); i++) {  
 n += FTI\_worker[i]->getSalary();  
 }  
  
 n = n / (FTI\_worker.size());  
 **return** n;  
 }  
  
 **int** vectorSize() {  
 **return** FTI\_worker.size();  
 }  
  
 Employee \*ptrWorker(**int** i) {  
 **return** FTI\_worker[i];  
 }  
  
 **void** eraseWorker(**int** i) {  
 FTI\_worker.erase(FTI\_worker.begin() + i);  
 }  
};

# Kib.h

#pragma once  
  
#include **"Division.h"  
  
class** Kib : **public** Division {  
**protected**:  
 vector<Employee \*> Kib\_worker;  
  
**public**:  
 Kib() : Division() {};  
  
 **void** add(Employee \*worker) override {  
 Kib\_worker.push\_back(worker);  
 }  
  
 **float** averageSalary() override {  
 **float** n;  
 **for** (**int** i = 0; i < Kib\_worker.size(); i++) {  
 n += Kib\_worker[i]->getSalary();  
 }  
 n = n / (Kib\_worker.size());  
 **return** n;  
 }  
  
 **int** vectorSize() {  
 **return** Kib\_worker.size();  
 }  
  
 Employee \*ptrWorker(**int** i) {  
 **return** Kib\_worker[i];  
 }  
  
 **void** eraseWorker(**int** i) {  
 Kib\_worker.erase(Kib\_worker.begin() + i);  
 }  
};

# MainMenu.cpp

#include **<iostream>**#include **<vector>**#include **<fstream>**#include **"Assistant.h"**#include **"Docent.h"**#include **"Engineer.h"**#include **"LabAssistant.h"**#include **"Professor.h"**#include **"SeniorTeacher.h"**#include **"IT.h"**#include **"Kib.h"**#include **"FTI.h"  
  
using namespace** std;  
  
**void** printSeparator() {  
 **for** (**int** i = 0; i < 100; i++)  
 cout << **"="**;  
 cout << endl;  
}  
  
**void** eraseWorker(vector<Employee \*> emp, IT &InfT, FTI &Ft, Kib &Ki, **int** num) {  
 **if** (emp[num - 1]->getDivision() == **"Institute of Information Technologies"**) {  
 **for** (**int** i = 0; i < InfT.vectorSize(); i++) {  
 **if** (emp[num - 1] == InfT.ptrWorker(i)) {  
 InfT.eraseWorker(i);  
 **break**;  
 }  
 }  
  
 } **else if** (emp[num - 1]->getDivision() == **"Institute of Physics and Technology"**)  
 **for** (**int** i = 0; i < Ft.vectorSize(); i++) {  
 **if** (emp[num - 1] == Ft.ptrWorker(i)) {  
 Ft.eraseWorker(i);  
 **break**;  
 }  
 }  
 **else  
 for** (**int** i = 0; i < Ki.vectorSize(); i++) {  
 **if** (emp[num - 1] == Ki.ptrWorker(i)) {  
 Ki.eraseWorker(i);  
 **break**;  
 }  
 }  
}  
  
**void** setDivision(vector<Employee \*> &emp, string buff, IT &InfT, FTI &Ft, Kib &Ki) {  
 **if** (buff == **"Institute of Information Technologies"**)  
 InfT.add(\*(emp.end() - 1));  
 **else if** (buff == **"Institute of Physics and Technology"**)  
 Ft.add(\*(emp.end() - 1));  
 **else** Ki.add(\*(emp.end() - 1));  
}  
  
**int** correctInput() {  
 string buff;  
 **int** n = 0;  
 **do** {  
 cout << **"Choice: "**;  
 cin >> buff;  
 n = 0;  
 **for** (**int** i = 0; i < buff.length(); i++) {  
 **if** ((buff[i] > 47) && (buff[i] < 58))  
 n++;  
 }  
 **if** ((n != buff.length()) || (buff[n] != **'\0'**)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 printSeparator();  
  
 } **while** ((n != buff.length()) || (buff[n] != **'\0'**));  
 **return** stoi(buff);  
  
}  
  
**int** getClasses(**int** sal, string pos) {  
 **if** (pos == **"Professor"**)  
 **return** sal / (250 \* 4 \* 5);  
 **if** (pos == **"Docent"**)  
 **return** sal / (250 \* 4 \* 4);  
 **if** (pos == **"Senior Teacher"**)  
 **return** sal / (250 \* 4 \* 3);  
 **if** (pos == **"Assistant"**)  
 **return** sal / (250 \* 4 \* 2);  
 **if** (pos == **"Engineer"**)  
 **return** sal / (125 \* 4 \* 5);  
 **if** (pos == **"Laboratory Assistant"**)  
 **return** sal / (125 \* 4 \* 4);  
 **return** 0;  
  
}  
  
  
**bool** readTxt(vector<Employee \*> &emp, IT &InfT, FTI &Ft, Kib &Ki) {  
 string buff1, buff2, buff3;  
 **int** buffInt1, buffInt2;  
 ifstream fin(**"..\\Employee.txt"**);  
 **if** (!(fin.is\_open())) {  
 fin.close();  
 ofstream{**"..\\Employee.txt"**, ios\_base::out};  
 fin.open(**"..\\Employee.txt"**);  
 }  
  
 **if** (!(fin.is\_open())) {  
 cout << **"File openning error!"** << endl;  
 **return true**;  
 }  
  
 fin.seekg(0, ios::end);  
 **if** (fin.tellg() == 0) {  
 fin.close();  
 **return false**;  
 }  
 fin.seekg(0, ios::beg);  
  
 **while** (!(fin.eof())) {  
  
  
 getline(fin, buff1, **'#'**);  
 fin >> buffInt1;  
 fin.ignore();  
 fin >> buffInt2;  
 fin.ignore();  
 getline(fin, buff2, **'#'**);  
 fin.ignore();  
 **switch** (buffInt2) {  
  
 **case** 1: {  
 emp.push\_back(**new** Professor(buff1, buff2, buffInt1));  
 **break**;  
 }  
 **case** 2: {  
 emp.push\_back(**new** Docent(buff1, buff2, buffInt1));  
 **break**;  
 }  
 **case** 3: {  
 emp.push\_back(**new** SeniorTeacher(buff1, buff2, buffInt1));  
 **break**;  
 }  
 **case** 4: {  
 emp.push\_back(**new** Assistant(buff1, buff2, buffInt1));  
 **break**;  
 }  
 **case** 5: {  
 emp.push\_back(**new** Engineer(buff1, buff2, buffInt1));  
 **break**;  
 }  
 **case** 6: {  
 emp.push\_back(**new** LabAssistant(buff1, buff2, buffInt1));  
 **break**;  
 }  
 }  
 setDivision(emp, buff2, InfT, Ft, Ki);  
  
 }  
 fin.close();  
 **return false**;  
  
}  
  
**void** printEmp(vector<Employee \*> emp) {  
 cout << **"List of the employees: "** << endl;  
 **for** (**int** i = 0; i < emp.size(); i++) {  
 cout << i + 1 << **". "** << emp[i]->getName() << endl;  
 }  
 printSeparator();  
}  
  
  
**void** completeOperation() {  
 cout << **"Action was successfully completed!"** << endl;  
 system(**"pause"**);  
 printSeparator();  
}  
  
**void** changePosition(vector<Employee \*> &emp, IT &InfT, FTI &Ft, Kib &Ki, **int** num) {  
 string buff1, buff2;  
 **int** buffInt, buffSwtch;  
 buff1 = emp[num - 1]->getName();  
 buff2 = emp[num - 1]->getDivision();  
 buffInt = getClasses(emp[num - 1]->getSalary(), emp[num - 1]->getPosition());  
  
  
 eraseWorker(emp, InfT, Ft, Ki, num);  
 swap(emp[num - 1], emp[emp.size() - 1]);  
 **delete** emp[emp.size() - 1];  
 emp.erase(emp.end() - 1);  
  
 cout << **"What kind of employee do you want to set?"** << endl;  
 cout << **"1. Teacher."** << endl;  
 cout << **"2. Teaching support staff."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 2)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 2));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 cout << **"What position do you want to set?"** << endl;  
 cout << **"1. Professor."** << endl;  
 cout << **"2. Docent."** << endl;  
 cout << **"3. Senior Teacher."** << endl;  
 cout << **"4. Assistant."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 4)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 4));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 emp.push\_back(**new** Professor(buff1, buffInt));  
 **break**;  
 }  
 **case** 2: {  
 emp.push\_back(**new** Docent(buff1, buffInt));  
 **break**;  
 }  
 **case** 3: {  
 emp.push\_back(**new** SeniorTeacher(buff1, buffInt));  
 **break**;  
 }  
 **case** 4: {  
 emp.push\_back(**new** Assistant(buff1, buffInt));  
 **break**;  
 }  
 }  
 **break**;  
 }  
 **case** 2: {  
 cout << **"What position do you want to set?"** << endl;  
 cout << **"1. Engineer."** << endl;  
 cout << **"2. Laboratory Assistant."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 2)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 2));  
  
 **switch** (buffSwtch) {  
 **case** 1: {  
 emp.push\_back(**new** Engineer(buff1, buffInt));  
 **break**;  
 }  
 **case** 2: {  
 emp.push\_back(**new** LabAssistant(buff1, buffInt));  
 **break**;  
 }  
 }  
 **break**;  
 }  
 }  
 **if** (buff2 == **"Institute of Information Technologies"**) {  
 InfT.add(\*(emp.end() - 1));  
 emp[emp.size() - 1]->setDivision(**"Institute of Information Technologies"**);  
 } **else if** (buff2 == **"Institute of Physics and Technology"**) {  
 Ft.add(\*(emp.end() - 1));  
 emp[emp.size() - 1]->setDivision(**"Institute of Physics and Technology"**);  
 } **else** {  
 Ki.add(\*(emp.end() - 1));  
 emp[emp.size() - 1]->setDivision(**"Institute of Cybernetics"**);  
 }  
  
  
 swap(emp[num - 1], emp[emp.size() - 1]);  
  
}  
  
  
**void** changeDivision(vector<Employee \*> emp, IT &InfT, FTI &Ft, Kib &Ki, **int** num) {  
 **int** buffSwtch;  
 eraseWorker(emp, InfT, Ft, Ki, num);  
 cout << **"What division do you want to set?"** << endl;  
 cout << **"1. Institute of Information Technologies."** << endl;  
 cout << **"2. Institute of Physics and Technology."** << endl;  
 cout << **"3. Institute of Cybernetics."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 3)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 3));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 InfT.add(\*(emp.begin() + (num - 1)));  
 emp[num - 1]->setDivision(**"Institute of Physics and Technology"**);  
 **break**;  
 }  
 **case** 2: {  
 Ft.add(\*(emp.begin() + (num - 1)));  
 emp[num - 1]->setDivision(**"Institute of Physics and Technology"**);  
 **break**;  
 }  
 **case** 3: {  
 Ki.add(\*(emp.begin() + (num - 1)));  
 emp[num - 1]->setDivision(**"Institute of Cybernetics"**);  
 **break**;  
 }  
 }  
  
  
}  
  
**void** deleteEmp(vector<Employee \*> &emp, IT &InfT, FTI &Ft, Kib &Ki, **int** num) {  
 eraseWorker(emp, InfT, Ft, Ki, num);  
 **delete** emp[num - 1];  
 emp.erase(emp.begin() + (num - 1));  
}  
  
**bool** isExistPos(vector<Employee \*> emp, string pos) {  
 **int** n = 0;  
 **for** (**int** i = 0; i < emp.size(); i++) {  
 **if** (emp[i]->getPosition() == pos)  
 n++;  
 }  
 **if** (n == 0)  
 **return false**;  
 **else  
 return true**;  
}  
  
  
**float** averageSalary(vector<Employee \*> emp, string pos) {  
 **int** n = 0;  
 **int** sal = 0;  
 **for** (**int** i = 0; i < emp.size(); i++) {  
 **if** (emp[i]->getPosition() == pos) {  
 n++;  
 sal += emp[i]->getSalary();  
 }  
 }  
 **return float**(sal) / n;  
}  
  
  
**void** writeTxt(vector<Employee \*> emp) {  
 ofstream fout(**"../Employee.txt"**);  
 **for** (**int** i = 0; i < emp.size(); i++) {  
 fout << emp[i]->getName() << **"#"** << emp[i]->getSalary() << **"#"**;  
 **if** (emp[i]->getPosition() == **"Professor"**)  
 fout << **"1"**;  
 **if** (emp[i]->getPosition() == **"Docent"**)  
 fout << **"2"**;  
 **if** (emp[i]->getPosition() == **"Senior Teacher"**)  
 fout << **"3"**;  
 **if** (emp[i]->getPosition() == **"Assistant"**)  
 fout << **"4"**;  
 **if** (emp[i]->getPosition() == **"Engineer"**)  
 fout << **"5"**;  
 **if** (emp[i]->getPosition() == **"Laboratory Assistant"**)  
 fout << **"6"**;  
 fout << **"#"** << emp[i]->getDivision() << **"#"**;  
 **if** (i != (emp.size() - 1))  
 fout << endl;  
  
 }  
 fout.close();  
}  
  
**void** deletePtr(vector<Employee \*> emp) {  
 **for** (**int** i = 0; i < emp.size(); i++)  
 **delete** emp[i];  
}  
  
  
**bool** isExist(vector<Employee \*> emp) {  
 **if** (emp.size() == 0)  
 **return false**;  
 **else  
 return true**;  
}

# main.cpp

#include **<vector>**#include **<fstream>**#include **"Assistant.h"**#include **"Docent.h"**#include **"Engineer.h"**#include **"LabAssistant.h"**#include **"Professor.h"**#include **"SeniorTeacher.h"**#include **"IT.h"**#include **"Kib.h"**#include **"FTI.h"  
  
void** printSeparator();  
**bool** readTxt(vector<Employee \*> &emp, IT &InfT, FTI &Ft, Kib &Ki);  
  
**void** printEmp(vector<Employee \*> emp);  
  
**void** completeOperation();  
  
**void** changePosition(vector<Employee \*> &emp, IT &InfT, FTI &Ft, Kib &Ki, **int** num);  
  
**void** setDivision(vector<Employee \*> &emp, string buff, IT &InfT, FTI &Ft, Kib &Ki);  
  
**int** getClasses(**int** sal, string pos);  
  
**void** changeDivision(vector<Employee \*> emp, IT &InfT, FTI &Ft, Kib &Ki, **int** num);  
  
**void** eraseWorker(vector<Employee \*> emp, IT &InfT, FTI &Ft, Kib &Ki, **int** num);  
  
**void** deleteEmp(vector<Employee \*> &emp, IT &InfT, FTI &Ft, Kib &Ki, **int** num);  
  
**float** averageSalary(vector<Employee \*> emp, string pos);  
  
**bool** writeTxt(vector<Employee \*> emp);  
  
**void** deletePtr(vector<Employee \*> emp);  
  
**bool** isExist(vector<Employee \*> emp);  
  
**bool** isExistPos(vector<Employee \*> emp, string pos);  
  
**int** correctInput();  
  
  
**int** main() {  
 string buff;  
 **int** buffInt, buffSal;  
 **int** buffSwtch;  
 vector<Employee \*> emp;  
 IT InfT;  
 FTI Ft;  
 Kib Ki;  
  
 **if** (readTxt(emp, InfT, Ft, Ki)) {  
 system(**"pause"**);  
 **return** 0;  
 }  
  
  
 **while** (**true**) {  
  
 cout << **"Welcome to the headcount management programm!"** << endl  
 << **"You can add, change or remove employees, print the list of employees or calculate the average salary for all kinds of employees or divisions."** << endl;  
  
 printSeparator();  
  
 cout << **"Available actions :"** << endl;  
 cout << **"1. Add employee."** << endl;  
 cout << **"2. Change employee."** << endl;  
 cout << **"3. Remove employee."** << endl;  
 cout << **"4. Print the full information of employees."** << endl;  
 cout << **"5. Calculate the average salary for all kinds of employees or divisions."** << endl;  
 cout << **"6. Exit the programm."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 6)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 6));  
  
 **switch** (buffSwtch) {  
 **case** 1: {  
 cout << **"Enter the name: "**;  
 cin.ignore();  
 getline(cin, buff);  
 printSeparator();  
 cout << **"Enter the count of working classes per week."** << endl;  
 buffInt = correctInput();  
 cout << **"What kind of employee do you want to add?"** << endl;  
 cout << **"1. Teacher."** << endl;  
 cout << **"2. Teaching support staff."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 2)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 2));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 cout << **"What position do you want to add?"** << endl;  
 cout << **"1. Professor."** << endl;  
 cout << **"2. Docent."** << endl;  
 cout << **"3. Senior Teacher."** << endl;  
 cout << **"4. Assistant."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 4)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 4));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 emp.push\_back(**new** Professor(buff, buffInt));  
 **break**;  
 }  
 **case** 2: {  
 emp.push\_back(**new** Docent(buff, buffInt));  
 **break**;  
 }  
 **case** 3: {  
 emp.push\_back(**new** SeniorTeacher(buff, buffInt));  
 **break**;  
 }  
 **case** 4: {  
 emp.push\_back(**new** Assistant(buff, buffInt));  
 **break**;  
 }  
 }  
 **break**;  
 }  
 **case** 2: {  
 cout << **"What position do you want to add?"** << endl;  
 cout << **"1. Engineer."** << endl;  
 cout << **"2. Laboratory Assistant."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 2)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 2));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 emp.push\_back(**new** Engineer(buff, buffInt));  
 **break**;  
 }  
 **case** 2: {  
 emp.push\_back(**new** LabAssistant(buff, buffInt));  
 **break**;  
 }  
 }  
 **break**;  
 }  
  
 }  
 cout << **"Available divisions: "** << endl;  
 cout << **"1. Institute of Information Technologies"** << endl;  
 cout << **"2. Institute of Physics and Technology"** << endl;  
 cout << **"3. Institute of Cybernetics"** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 3)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 3));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 emp[emp.size() - 1]->setDivision(**"Institute of Information Technologies"**);  
 InfT.add(\*(emp.end() - 1));  
 **break**;  
 }  
 **case** 2: {  
 emp[emp.size() - 1]->setDivision(**"Institute of Physics and Technology"**);  
 Ft.add(\*(emp.end() - 1));  
 **break**;  
 }  
 **case** 3: {  
 emp[emp.size() - 1]->setDivision(**"Institute of Cybernetics"**);  
 Ki.add(\*(emp.end() - 1));  
 **break**;  
 }  
 }  
 completeOperation();  
 **break**;  
 }  
 **case** 2: {  
 **if** (!(isExist(emp))) {  
 cout << **"There aren't any employees!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 printEmp(emp);  
 cout << **"Choose employee."** << endl;  
 **do** {  
 buffInt = correctInput();  
 **if** ((buffInt < 1) || (buffInt > emp.size())) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffInt < 1) || (buffInt > emp.size()));  
 cout << **"Available actions: "** << endl;  
 cout << **"1. Change name."** << endl;  
 cout << **"2. Change salary."** << endl;  
 cout << **"3. Change position."** << endl;  
 cout << **"4. Change division."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 4)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 4));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 cin.ignore();  
 cout << **"Enter the new name: "**;  
 getline(cin, buff);  
 emp[buffInt - 1]->setName(buff);  
 completeOperation();  
 **break**;  
 }  
 **case** 2: {  
 cout << **"Enter the new count of working classes per week."** << endl;  
 buffSal = correctInput();  
 emp[buffInt - 1]->setSalary(buffSal);  
 completeOperation();  
 **break**;  
 }  
 **case** 3: {  
 changePosition(emp, InfT, Ft, Ki, buffInt);  
 completeOperation();  
 **break**;  
 }  
  
 **case** 4: {  
 changeDivision(emp, InfT, Ft, Ki, buffInt);  
 completeOperation();  
 **break**;  
 }  
 }  
 **break**;  
  
 }  
 **case** 3: {  
 **if** (!(isExist(emp))) {  
 cout << **"There aren't any employees!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 printEmp(emp);  
 cout << **"Choose employee."** << endl;  
 **do** {  
 buffInt = correctInput();  
 **if** ((buffInt < 1) || (buffInt > emp.size())) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffInt < 1) || (buffInt > emp.size()));  
 deleteEmp(emp, InfT, Ft, Ki, buffInt);  
 completeOperation();  
 **break**;  
 }  
 **case** 4: {  
 **if** (!(isExist(emp))) {  
 cout << **"There aren't any employees!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 printEmp(emp);  
 cout << **"Choose employee."** << endl;  
 **do** {  
 buffInt = correctInput();  
 **if** ((buffInt < 1) || (buffInt > emp.size())) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffInt < 1) || (buffInt > emp.size()));  
 emp[buffInt - 1]->print();  
 printSeparator();  
 completeOperation();  
 **break**;  
 }  
 **case** 5: {  
 **if** (!(isExist(emp))) {  
 cout << **"There aren't any employees!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"1. Average salary for employees."** << endl;  
 cout << **"2. Average salary for divisions."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 2)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 2));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 cout << **"1. Average salary for professors."** << endl;  
 cout << **"2. Average salary for docents."** << endl;  
 cout << **"3. Average salary for senior teachers."** << endl;  
 cout << **"4. Average salary for assistants."** << endl;  
 cout << **"5. Average salary for engineers."** << endl;  
 cout << **"6. Average salary for laboratory assistants."** << endl;  
 cout << **"7. Average salary for all employees."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 7)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 7));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 buff = **"Professor"**;  
 **if** (!(isExistPos(emp, buff))) {  
 cout << **"There aren't any "** << buff << **"s in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary: "** << averageSalary(emp, buff) << endl;  
 completeOperation();  
 **break**;  
 }  
 **case** 2: {  
 buff = **"Docent"**;  
 **if** (!(isExistPos(emp, buff))) {  
 cout << **"There aren't any "** << buff << **"s in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary: "** << averageSalary(emp, buff) << endl;  
 completeOperation();  
 **break**;  
 }  
 **case** 3: {  
 buff = **"Senior Teacher"**;  
 **if** (!(isExistPos(emp, buff))) {  
 cout << **"There aren't any "** << buff << **"s in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary: "** << averageSalary(emp, buff) << endl;  
 completeOperation();  
 **break**;  
 }  
 **case** 4: {  
 buff = **"Assistant"**;  
 **if** (!(isExistPos(emp, buff))) {  
 cout << **"There aren't any "** << buff << **"s in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary: "** << averageSalary(emp, buff) << endl;  
 completeOperation();  
 **break**;  
 }  
 **case** 5: {  
 buff = **"Engineer"**;  
 **if** (!(isExistPos(emp, buff))) {  
 cout << **"There aren't any "** << buff << **"s in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary: "** << averageSalary(emp, buff) << endl;  
 completeOperation();  
 **break**;  
 }  
 **case** 6: {  
 buff = **"Laboratory Assistant"**;  
 **if** (!(isExistPos(emp, buff))) {  
 cout << **"There aren't any "** << buff << **"s in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary: "** << averageSalary(emp, buff) << endl;  
 completeOperation();  
 **break**;  
 }  
 **case** 7: {  
 buffSal = 0;  
 **for** (**int** i = 0; i < emp.size(); i++)  
 buffSal += emp[i]->getSalary();  
 cout << **"Average salary: "** << **float**(buffSal) / emp.size() << endl;  
 completeOperation();  
 **break**;  
 }  
 }  
  
 **break**;  
 }  
 **case** 2: {  
 cout << **"1. Average salary for Institute of Inforamation Technologies."** << endl;  
 cout << **"2. Average salary for Institute of Physics and Technology."** << endl;  
 cout << **"3. Average salary for Institute of Cybernetics."** << endl;  
 printSeparator();  
 **do** {  
 buffSwtch = correctInput();  
 **if** ((buffSwtch < 1) || (buffSwtch > 3)) {  
 cout << **"Invalid input! Try again."** << endl;  
 }  
 } **while** ((buffSwtch < 1) || (buffSwtch > 3));  
 **switch** (buffSwtch) {  
 **case** 1: {  
 **if** (InfT.vectorSize() == 0) {  
 cout << **"There aren't any IT workers in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary : "** << InfT.averageSalary() << endl;  
 completeOperation();  
 **break**;  
 }  
 **case** 2: {  
 **if** (Ft.vectorSize() == 0) {  
 cout << **"There aren't any FTI workers in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary : "** << Ft.averageSalary() << endl;  
 completeOperation();  
 **break**;  
 }  
 **case** 3: {  
 **if** (Ki.vectorSize() == 0) {  
 cout << **"There aren't any Kib workers in the employee list!"** << endl;  
 printSeparator();  
 **break**;  
 }  
 cout << **"Average salary : "** << Ki.averageSalary() << endl;  
 completeOperation();  
 **break**;  
 }  
 }  
 **break**;  
 }  
 }  
 **break**;  
 }  
  
  
 **case** 6: {  
 writeTxt(emp);  
 deletePtr(emp);  
  
 **return** 0;  
 }  
  
 }  
 }  
 system(**"pause"**);  
 **return** 0;  
}

**ОТЗЫВ РУКОВОДИТЕЛЯ КУРСОВОЙ РАБОТЫ (КР/КП)**

**ПО КАФЕДРЕ ИиППО ИНСТИТУТА ИТ МИРЭА**

Настоящий отзыв составлен руководителем КР/КП студента **Тимошкина Максима Сергеевича, ИКБО-13-18, 18И0502** по итогам выполнения им на кафедре ИиППО института ИТ **Курсовой работы** по дисциплине объектно-ориентированное программирование в 2 семестре 1 курса обучения в академическом бакалавриате на основании ФГОС ВО и действующих в РТУ МИРЭА нормативных актов: «Инструкция по организации и проведению курсового проектирования (СМКО МИРЭА 7.5.1/04.И.05)»; «Временное положение о текущем контроле успеваемости и промежуточной аттестации по образовательным программам высшего образования – программам бакалавриата, программам специалитета и программам магистратуры (Положение. СМКО МИРЭА 7.5.1/03.П.08)»; «Рекомендации по оформлению письменных работ обучающимися (СМКО МИРЭА 7.5.1/03.П.69)»; а также Рабочей программы поименованной дисциплины.

Руководителем КР/КП отмечается, что задание на курсовую работу выдано в первые две недели семестра. В это же время со студентом проведено собеседование по вопросам организации, учебно-трудовой дисциплины, графика работ и взаимодействия с руководителем, поставлены, обеспечены информационными ресурсами и обсуждены цель, основная задача, содержание, технологии, информационно-методическое обеспечение, формы и сущности отчётности и её защиты под дифференцированную аттестацию. Проведён инструктаж по соблюдению требований охраны труда, техники безопасности (ТБ), а также правилам внутреннего трудового (учебного) распорядка (ПВР).

За студентом закреплен тьютор-стажёр из числа актива подразделения СНК при кафедре ИиППО, выход в кафедральный тематический мини-портал Кабинета дипломного проектирования и магистерской подготовки при кафедре ИиППО (далее: «Кабинет»); в портале создана соответствующая директория курсовой работы, в распоряжение проектанта предоставлены стартовые информационные консисты тематических библиотек портала, выданы методические указания и типовые шаблоны документального оформления пояснительной записки (ПЗ), относящиеся к конкретной дисциплине – в совокупности позволяющие эффективно использовать средства Онтонет управления образовательным макромедиа контентом, а также всемирных технологий WorkBooks. Студент ознакомлен с ТЗ, задачами, технологическими особенностями, применяемым ПО, сущностью этапа выполнения кафедральных НИР, грантов, текущих научно-информационных-технологических работ по развитию информационно-технологической базы кафедры.

Обозначено и реализовано обретение обучающимся профессиональных компетенций, а именно ОПК-1, ПК-14 и ПК-20.

Отчётные материалы студента в целом отвечают заданной теме, поставленной задаче, требованиям по оформлению. Предложенные решения обладают полнотой и качественной глубиной раскрытия изучаемых вопросов, демонстрируя признаки новизны, креативности и меры самостоятельности, а также проявление проектантом дисциплинированности, элементов профессиональной этики, умения работать в коллективе.

Качество, полнота и технико-эстетический уровень оформления ПЗ удовлетворяют типовым требованиям. Проверенная руководителем на работоспособность, соответствие и незашумлённость электронная версия ПЗ в форматах Word и Pdf удовлетворительна.

По итогам курсовой работы, представления ПЗ руководитель согласно перечисленным выше нормативам подводит следующие итоги:

1. Качественная сторона выполненных работ - \_\_\_\_\_\_\_\_\_\_\_\_*(отл., хор., удовл., неудовл.)*.
2. Количественная сторона (характеризующая охват, объём, детализацию и т.п.) выполненных работ \_\_\_\_\_\_\_\_ *(отл., хор., удовл., неудовл.)*.
3. Содержание планово-отчётной документации проектанта\_\_\_\_\_\_\_\_ *(отл., хор., удовл., неудовл.)*.
4. Качество оформления документации на бумажном носителе и в электронной форме (в том числе, соблюдение требований стандартов и упомянутых выше нормативов) \_\_\_\_\_\_\_\_\_\_\_*(отл., хор., удовл., неудовл.)*.
5. Учебно-производственная дисциплина, включая соблюдение установленных сроков выполнения, отчётности и аттестаций, а также норм учебно-профессиональной этики \_\_\_\_\_\_\_\_\_*(отл., хор., удовл., неудовл.)*.

Итоговая оценка, рекомендованная руководителем:\_\_\_\_\_\_\_\_\_\_\_\_ *(отл., хор., удовл., неудовл.).*

Таким образом, курсовая работа успешно и в срок завершено. Курсовая работа рекомендована к защите.

Руководитель:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(должность, ф.и.о., дата)

**Прим.:** «Кабинет» каф. ИиППО м-лы КП / КР на хранение принял»

Подпись отв. лица. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Должность. Дата.