Министерство связи и информатизации Республики Беларусь
Учреждение образования
БЕЛОРУССКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ СВЯЗИ
Факультет электросвязи
Кафедра программного обеспечения сетей телекоммуникаций

РАЗРАБОТКА СИСТЕМЫ УЧЕТА ВЫДАННЫХ ЗАДАНИЙ

Пояснительная записка
к курсовому проекту
по дисциплине
«Объектно-ориентированное программирование»

Выполнил студент гр. СП641 Д. С. Бабанин
Руководитель Т. Л. Труханович

Минск 2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 4
2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 5
2.1 Функциональные требования 5
2.2 Диаграмма вариантов использования и ее описание 6
2.3 Диаграммы действий 13
3 ОПИСАНИЕ ПРОГРАММНОГО КОДА 26
3.1 Архитектура программы 27
3.2 Описание диаграммы классов 27
3.2.1 Класс Date 28
3.3.2 Класс Task 29
3.2.3 Класс UserAccount 29
3.2.4 Класс ServiceTasks 33
3.2.5 Класс ServiceUserAccounts 36
3.2.6 Класс LoginAndPasswordScreen 17
3.2.7 Класс AdministratorScreen 38
3.2.8 Класс UserScreen 39
3.2.9 Класс HashPasswordWithSal 39
3.3 Описание функций, которые не вошли в диаграмму классов 39
3.3.1 Описание функций из cpp-файла main.cpp 39
3.3.2 Описание функций из head-файл ExceptionScreen.h 40
3.3.3 Описание функций из head-файл ServiceData.h 40
3.3.4 Описание функций из head-файл ControllerIOFStream.h 40
3.3.5 Описание функций из head-файл ControllerIStream.h 41
4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 42
5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 56
ЗАКЛЮЧЕНЕ 71
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 72
Приложение А 73
Приложение Б 74
Приложение В 75

ВВЕДЕНИЕ

Задачами курсового проекта по предмету «Объектно-ориентированное программирование» являются:

– приобретение теоретических знаний в области анализа систем средней сложности, построения информационных систем различной архитектуры;

– практическая работа по проектированию и реализации информационных систем.

Согласно заданию курсового проекта, необходимо разработать программу для просмотра сведений о сотрудниках предприятия с возможность выдачи и контроля выполнения заданий выданных сотрудникам предприятия.

Поставленная задача актуальна, поскольку на многих предприятиях количество сотрудников составляет пару тысяч и пересчитать всех в ручную довольно трудоемкий процесс, тем более контролировать их рабочий процесс. Поэтому решение данной задачи будет очень востребовано на крупных предприятиях с большим количеством сотрудников.

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

В данное время очень много предприятий с большим количеством сотрудников, при этом людей которые следят за выполнением заданий сотрудниками не так уж и много, в следствии чего, в момент, когда начинается контроль выполнения заданий, очень большая нагрузка ложится на этих людей, и происходят ошибки. Для того что бы избежать всех неудобств и будет разработана программа для большого количества сотрудников и контроля выполнения заданий.

Исходные данные к выполнению курсового проекта:

В отделе имеются сотрудники, которым руководитель выдает задания. Каждому сотруднику руководитель может выдать одно или несколько заданий. Запись в журнале выданных заданий содержит: код проекта к которому относится задание; содержание задания, код сотрудника, которому выдано задание, дата выдачи задания, планируемая дата сдачи задания; реальная дата сдачи задания; примечание. Вывести список заданий, до истечения срока выполнения которых осталось 3 дня; просроченные задания; список заданий у заданного сотрудника.

В качестве функциональных требований, стоит отметить навигацию между различными меню, сортировку, вывод, редактирование и добавление информации, а также добавление новых пользователей и редактирование профилей старых.

2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

2.1 Функциональные требования

В результате должен быть разработан комплекс программных средств, позволяющих осуществлять обработку данных согласно варианту задания.

Программное обеспечение основано на работе с файлами в консольном режиме.

Внизу программных форм должна выводиться информация о разработчике, руководителе проекта, год разработки. Предусмотреть возможность вставки на страницу текущей даты и времени.

Комплекс программных средств включает функционала: «Администратор» (полный доступ ко всему функционалу программы) и “Пользовательский» (ограниченный доступ к функционалу программы.

Программа «Администратор» должна обеспечивать выполнение следующих функциональных возможностей:

1. Управление учетными записями пользователей:
   – просмотреть все учетные записи;
   – добавить учетную запись;
   – отредактировать учетную запись;
   – удалить учетную запись.
2. Работа с данными:
   – режим редактирования:
   – просмотреть все данные;
   – добавить запись;
   – удалить запись (для этого необходимо ввести порядковый номер конкретной записи);
   – редактировать запись (для этого необходимо ввести порядковый номер конкретной записи);

Программа «Пользователь» должна обеспечивать выполнение следующих функциональных возможностей:

– просмотреть все данные;
– выполнить задачу (задачи), указанную в индивидуальном задании;
– выполнить поиск данных;
– выполнить сортировку по различным полям в алфавитном порядке / в порядке убывания.

Работа программы начинается с авторизации. Данные об учетных записях пользователей хранятся в файле следующей структуры:

– login;
– password;

Предусмотреть возможность шифрования паролей пользователей.

После прохождения авторизации на экран выводится меню.

После авторизации пользователю должно быть выведено меню, из которого, путем ввода номера соответствующего пункта меню, пользователь

переходит к выполнению нужной ему операции. Далее происходит обработка данных.

Предусмотреть:

– обработку исключительных ситуаций (ввод некорректных данных);

– возможность возврата назад (навигация);

– в случае отсутствия результатов поиска должно быть выведено сообщение «По Вашему запросу ничего не найдено.».

– запрос на подтверждение удаления вида «Вы действительно хотите удалить файл (запись)?»;

– вывод сообщения о результате выполнения каждой из операций (например, «Запись успешно создана»).

– работа меню должна быть реализована в виде класса.

2.2 Диаграмма вариантов использования и ее описание

Диаграмма вариантов использования в данном курсовом проекте приложении предоставлена в приложении А. На диаграмме, действующие субъекты предоставляются человечками, варианты использования – эллипсами. Прямоугольная рамка окружает все варианты использования, оставляя за своими пределами действующих субъектов, формируя границу системы. Внутри границы системы находится программное обеспечение, которое необходимо создать.

Начало программы:

Отображается приветственное окно. Программа загружает из файла данные об учетных записях пользователей и заданий. На экране отображается форма для ввода логина и пароля.

Меню (админ):

На экране появляется меню администратора, в котором он может выбрать с какими данными он хочет работать:

1. Учетные записи пользователей
2. Задания и статистика
3. Сохранение изменений
0. Выйти

Меню управления записями пользователя (админ):

На экране появляется меню для работы с учетными записями пользователей, для выбора пункта меню необходимо ввести соответствующую цифру:

Список учетных записей пользователей

1. Добавить учетную запись пользователя
2. Редактировать учетную запись пользователя
3. Удаление учетной записи пользователя
4. Удаление всех учетных записей пользователей
5. Сортировка по возрастанию/А-Я (по заданному полю)
6. Сортировка по убыванию/Я-А (по заданному полю)
7. Поиск
0. Вернуться

Меню управления заданиями (админ):

На экране появляется меню работы с заданиями, для выбора пункта меню необходимо ввести соответствующую цифру:

Список заданий

1. Добавить задание
2. Редактировать задание
3. Удаление задания
4. Удаление всех записей
5. Сортировка по возрастанию/А-Я (по заданному полю)
6. Сортировка по убыванию/Я-А (по заданному полю)
7. Поиск – строгий, мягкий, в начале строки, в конце строки, с учетом регистра, без учета регистра
8. Список просроченных заданий
9. Список заданий у заданного сотрудника
10. Список заданий, до истечения срока выполнения которых осталось X дней
0. Назад

Меню управления заданиями и статистика (пользователь):

На экране появляется меню работы с заданиями, для выбора пункта меню необходимо ввести соответствующую цифру:

Список заданий (пользователя)

1. Уведомить о выполнении задания
2. Сортировка по убыванию/Я-А (по заданному полю)
3. Сортировка по возрастанию/А-Я (по заданному полю)
4. Поиск
5. Список просроченных заданий
6. Список заданий, до истечения срока выполнения которых осталось X дней
0. Выйти и сохранить изменения

Добавить учетную запись пользователя (админ):

Генерируется идентификатор UserId. На экран отображается сообщение, в котором программа просит ввести данные о пользователе и заранее показывает UserId пользователя, к которому эти данные будут приписаны:

- Пароль
- Фамилия
- Имя
- Отчество
- Отдел

Программа требует подтверждения о добавлении учетной записи пользователя. Пароль шифруется. Информация заноситься в список учетных записей пользователей и сортируется по Фамилиям сотрудников.

Редактировать учетную запись пользователя (админ):

На экран отображается сообщение, в котором программа просит ввести UserId учетной записи пользователя, данные которого необходимо изменить. После программа отображает сообщение с данными об учетной записи пользователя, в котором программа просит подтверждение о редактировании этой учетной записи пользователя. После на экран отобразиться сообщение, в котором программа просит выбрать поля учетной записи пользователя, которое вы хотите изменить:

- Фамилию Имя Отчество учетной записи пользователя
- Отдел учетной записи пользователя
- Пароль учетной записи пользователя (для его изменения необходимо знать старый пароль)
- Все данные об учетной записи пользователя (кроме UserId)

Пароль шифруется. Информация заменяется в списке учетных записей пользователей и сортируется по Фамилиям учетных записей пользователей.

Удаление учетной записи пользователя (админ):

На экран отображается сообщение, в котором программа просит ввести UserId учетной записи пользователя, данные которого необходимо удалить. После программа отображает сообщение с данными об учетной записи пользователя, в котором программа просит подтверждение об удалении этой учетной записи пользователя. После подтверждения программа удаляет пользователя из списка вместе с его заданиями в другом списке.

Удаление всех учетных записей пользователей (админ):

На экран отображается сообщение, в котором программа просит подтверждения об удалении всех учетных записей пользователей. После подтверждения программа удаляет всех пользователей и чистит весь список заданий.

Сортировка по убыванию/Я-А (админ):

На экран отображается сообщение, в котором программа прости ввести номер того поля, по которому вы хотите отсортировать:

1. По идентификатору
2. По фамилии
3. По имени
4. По отчеству
5. По отделу

После программа сортирует по убыванию список учетных записей пользователей по заданному полю. На экран отображает меню управления учетными записями пользователей с отсортированным списком учетных записей пользователей.

Сортировка по возрастанию/А-Я (по заданному полю) (админ):

На экран отображается сообщение, в котором программа прости ввести номер того поля, по которому вы хотите отсортировать:

1. По идентификатору
2. По фамилии
3. По имени
4. По отчеству
5. По отделу

После программа сортирует по возростанию список учетных записей пользователей по заданному полю. На экран отображает меню управления учетными записями пользователей с отсортированным списком учетных записей пользователей.

Поиск (админ):

На экран отображается сообщение, в которое программа прости ввести номер того поля, по которому будет производиться поиск:

1. По идентификатору
2. По фамилии
3. По имени
4. По отчеству
5. По отделу
0. Назад

После на экран отображается сообщение, в котором программа просит ввести данные для поиска учетной записи пользователя. После программа создает новый список, в котором будет хранить результаты поиска. Программа совершает поиск и нужные записи заносит в ранее созданный список. После на экран отображается список учётных записей пользователей с найденными записями.

Добавить задание (админ):

Генерируется идентификатор TaskId. На экран отображается сообщение, в котором программа простит ввести данные о задании и заранее показывает идентификатор TaskId, которое будет приписано к этому заданию:

- Задание
- Идентификатор пользователя, которому принадлежит задание
- Дата выдачи задания
- Планируемая дата сдачи
- Дата сдачи задания
- Примечание

После программа проверяет, есть ли такой идентификатор пользователя, который соответствует введенным данным. Если есть, то запись сохраняется в список заданий и список сортируется по UserId.

Редактировать задание (строгий поиск):

На экран отображается сообщение, в котором программа просит ввести TaskId задания. После на экран отображается сообщение с данными о задании, в котором программа просит ввести подтверждение об изменении задания. После на экран отображается сообщение, в котором программа простит выбрать поле, которое необходимо изменить:

- Задание
- Идентификатор пользователя, которому принадлежит задание
- Дата выдачи задания
- Планируемая дата сдачи
- Дата сдачи задания
- Примечание
- Все поля

После программа проверяет UserId. На экран отображается сообщение об успешном редактировании задания.

Внимание: в случае пользователя программа допускает изменения только в дате выполнения задания, но если задание просрочено, то он не способен выполнить это действие и должен обратиться к админу.

Удаление задания (админ):

На экран отображается сообщение, в котором программа просит ввести TaskId задания. После на экран отображается сообщение с данными о задании, в котором программа просит ввести подтверждение об удалении задания. После на экран отображается сообщение об успешном удалении задания.

Удаление всех заданий (админ):

На экран отображается сообщение, в котором программа простит ввести подтверждение об удалении всех заданий. После программа отображает подтверждение об удалении всех заданий.

Сортировка по убыванию/Я-А:

На экран отображается сообщение, в котором программа прости ввести номер того поля, по которому вы хотите отсортировать:

1. Идентификатор задания
2. Задание
3. Идентификатор пользователя, которому принадлежит задание
4. Дата выдачи задания
5. Планируемая дата сдачи
6. Дата сдачи задания
7. Примечание

После программа сортирует по убыванию список заданий по заданному полю. На экран отображает меню управления заданиями с отсортированным списком заданий.

Внимание: в случае пользователя будут сортироваться только список его заданий.

Сортировка по убыванию/А-Я (по заданному полю):

На экран отображается сообщение, в котором программа прости ввести номер того поля, по которому вы хотите отсортировать:

1. Идентификатор задания
2. Задание
3. Идентификатор пользователя, которому принадлежит задание
4. Дата выдачи задания
5. Планируемая дата сдачи
6. Дата сдачи задания
7. Примечание

После программа сортирует по возрастанию список заданий по заданному полю. На экран отображает меню управления заданиями с отсортированным списком заданий.

Внимание: в случае пользователя будут сортироваться только список его заданий.

Поиск:

После на экран отображается сообщение, в которое программа прости ввести номер того поля, по которому будет производиться поиск:

1. Идентификатор задания
2. Задание
3. Идентификатор пользователя, которому принадлежит задание
4. Дата выдачи задания
5. Планируемая дата сдачи
6. Дата сдачи задания
7. Примечание

После на экран отображается сообщение, в котором программа просит ввести данные для поиска задания. После программа создает новый список, в котором будет хранить результаты поиска. Программа совершает поиск и нужные записи заносит в ранее созданный список. После на экран отображается список заданий с найденными записями.

Внимание: в случае пользователя происходить поиск только его заданий.

Список просроченных заданий:

Программа создает новый список, выполняет поиск просроченных заданий и заносит их в этот список. На экран отображается сообщение, в котором программа отображает список просроченных заданий.

Внимание: в случае пользователя происходить поиск только его заданий.

Список заданий у заданного сотрудника (админ):

На экран отображается сообщение, в котором программа просит ввести UserId. Программа создает новый список, выполняет поиск заданий и заносит их в этот список. На экран отображается сообщение, в котором программа отображает список заданий конкретного пользователя.

Список заданий, до истечения срока выполнения которых осталось 3 дней:

На экран отображается сообщение, в котором программа отображает список заданий до истечения срока выполнения которых осталось 3 дней.

Внимание: в случае пользователя происходить поиск только его заданий.

Сохранение изменений:

На экран выводиться сообщение, в котором программа прости подтвердить сохранение изменений об учетных записях пользователей и заданиях в специальные файлы.

2.3 Диаграммы действий

Смоделируем варианты использования программы на примере диаграмм действий.

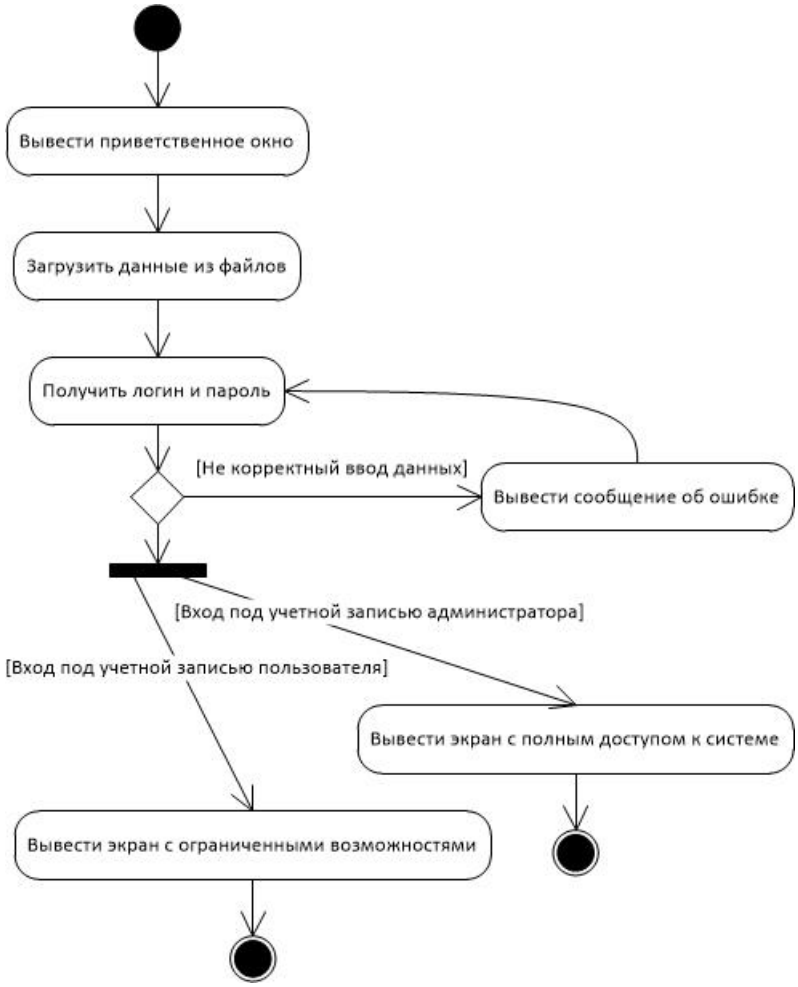

Рисунок 2.1 – Начало программы

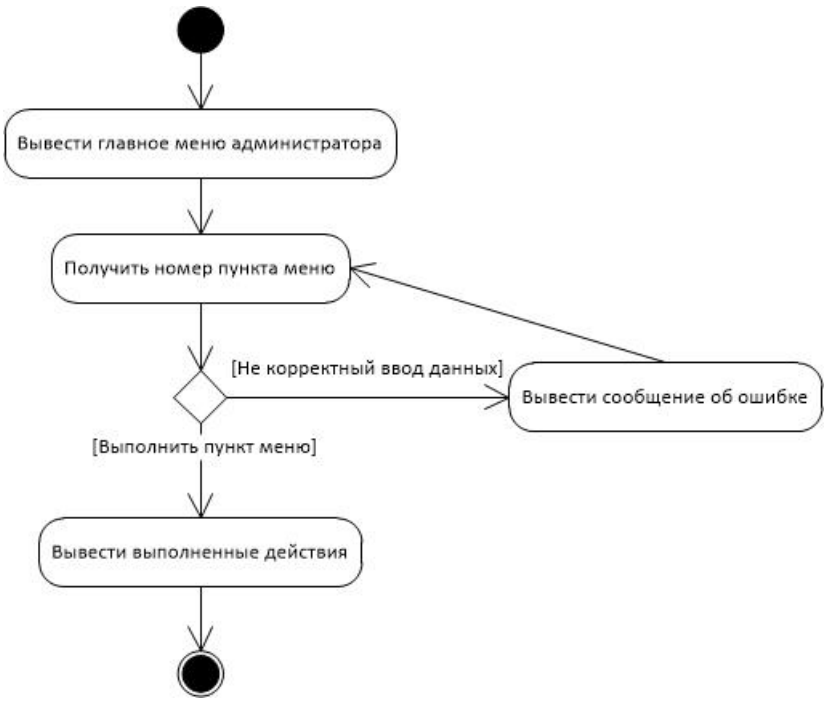

Рисунок 2.2 – Главное меню администратора

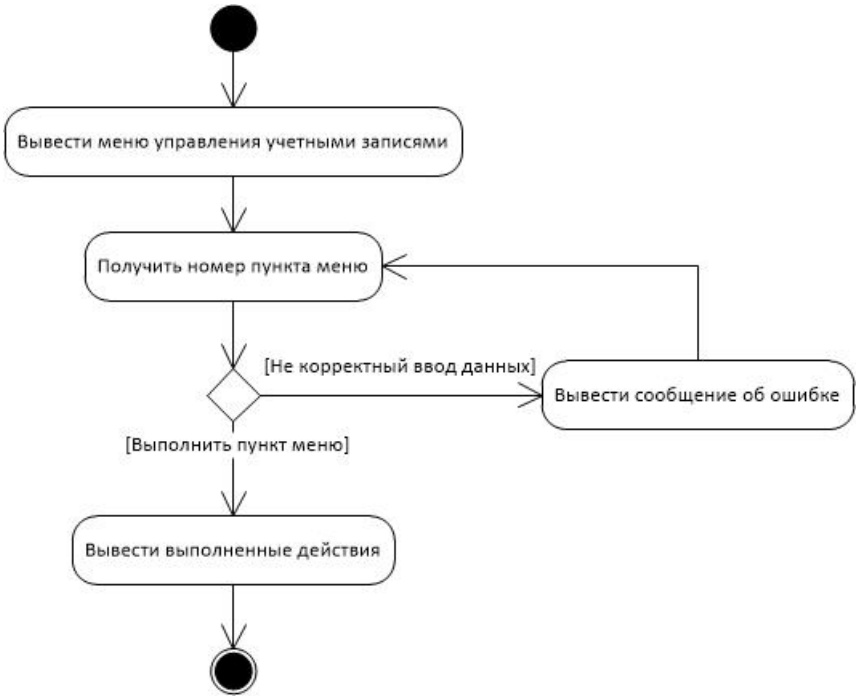

Рисунок 2.3 – Меню управления учетными записями пользователей

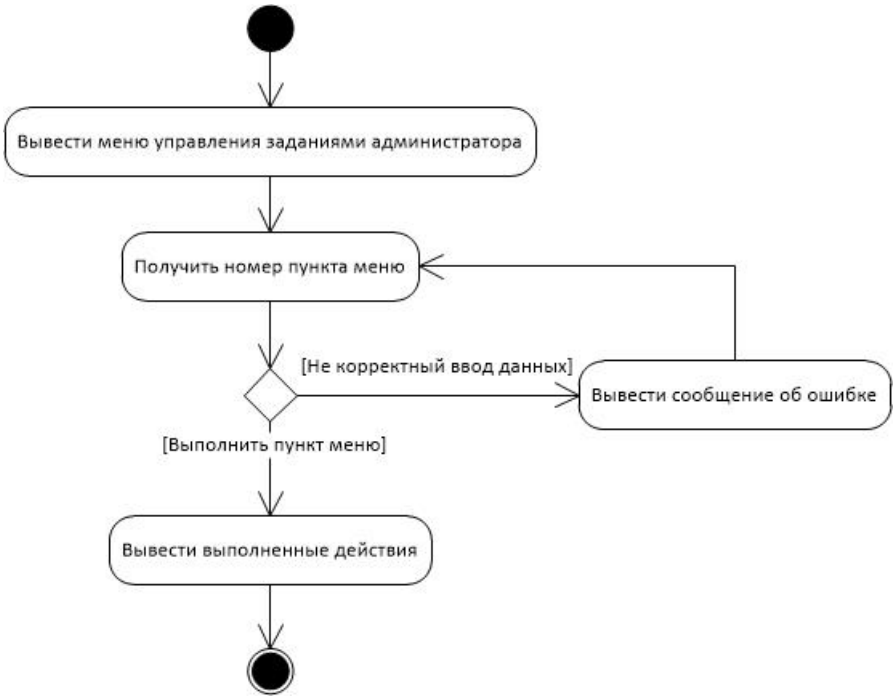

Рисунок 2.4 – Меню управления заданиями администратора

Рисунок 2.5 – Создание новой учетной записи

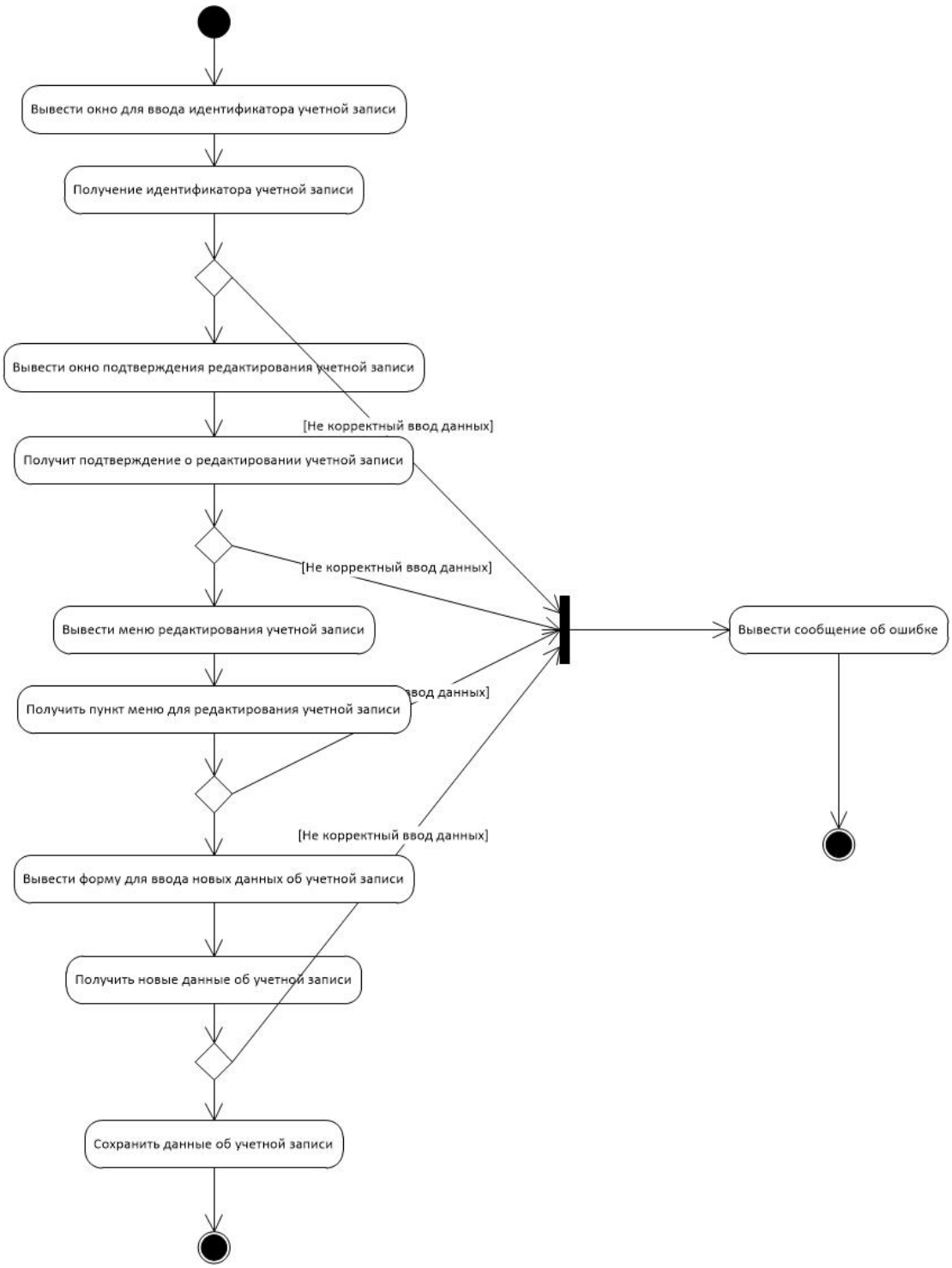

Рисунок 2.6 – Редактирование учетной записи

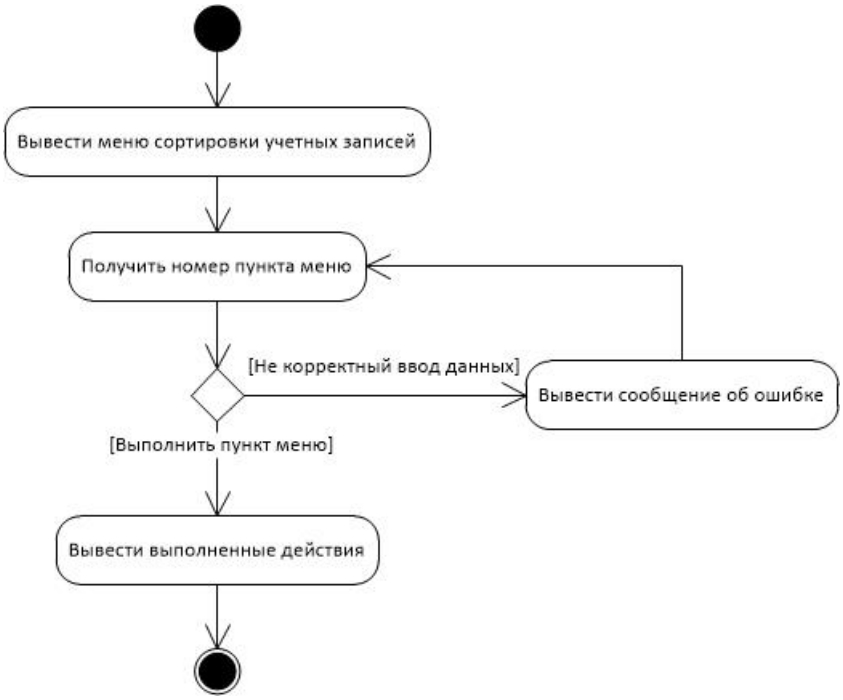

Рисунок 2.7 – Меню сортировки учетных записей

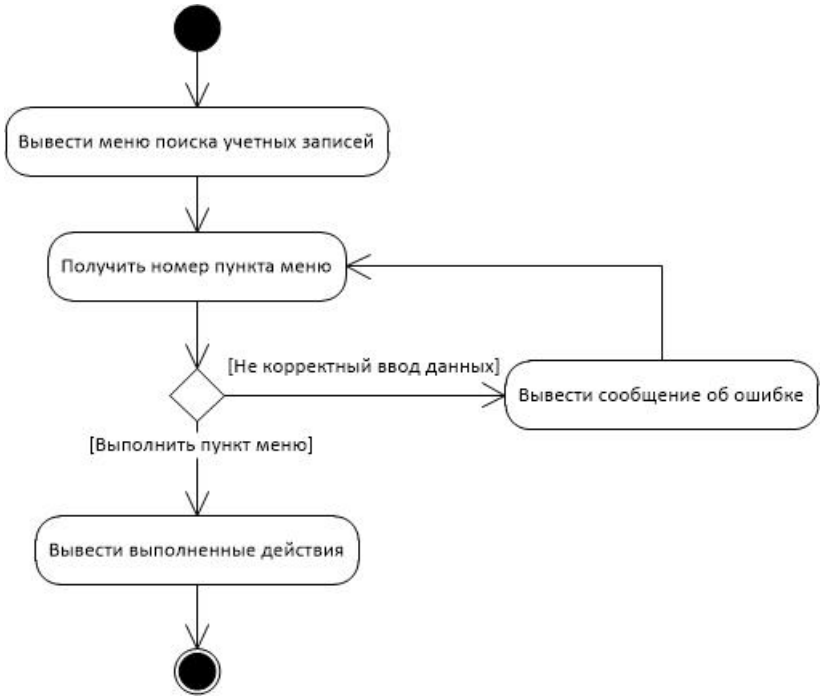

Рисунок 2.8 – Меню поиска учетных записей

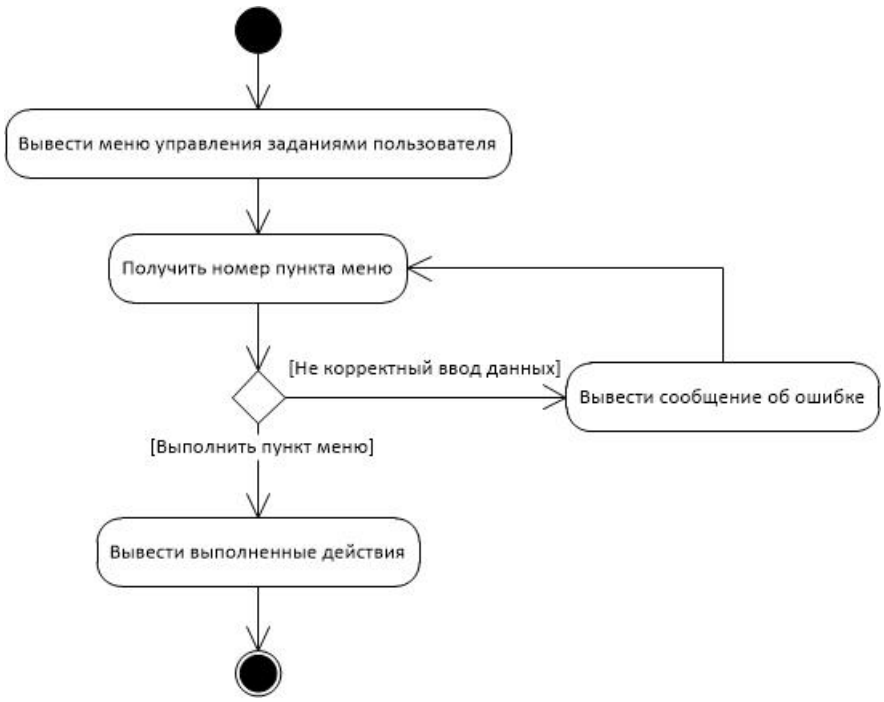

Рисунок 2.9 – Меню управления заданиями пользователя

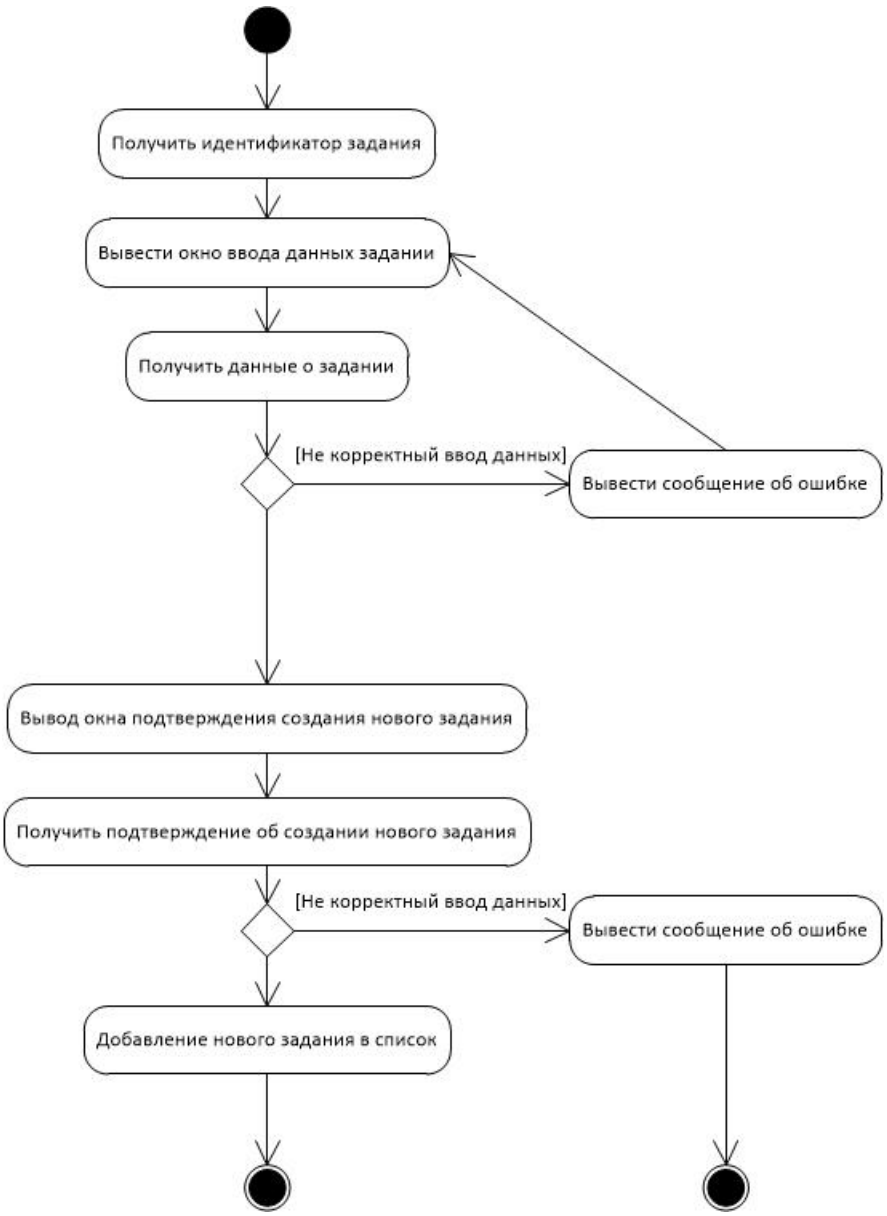

Рисунок 2.10 – Создание нового задания

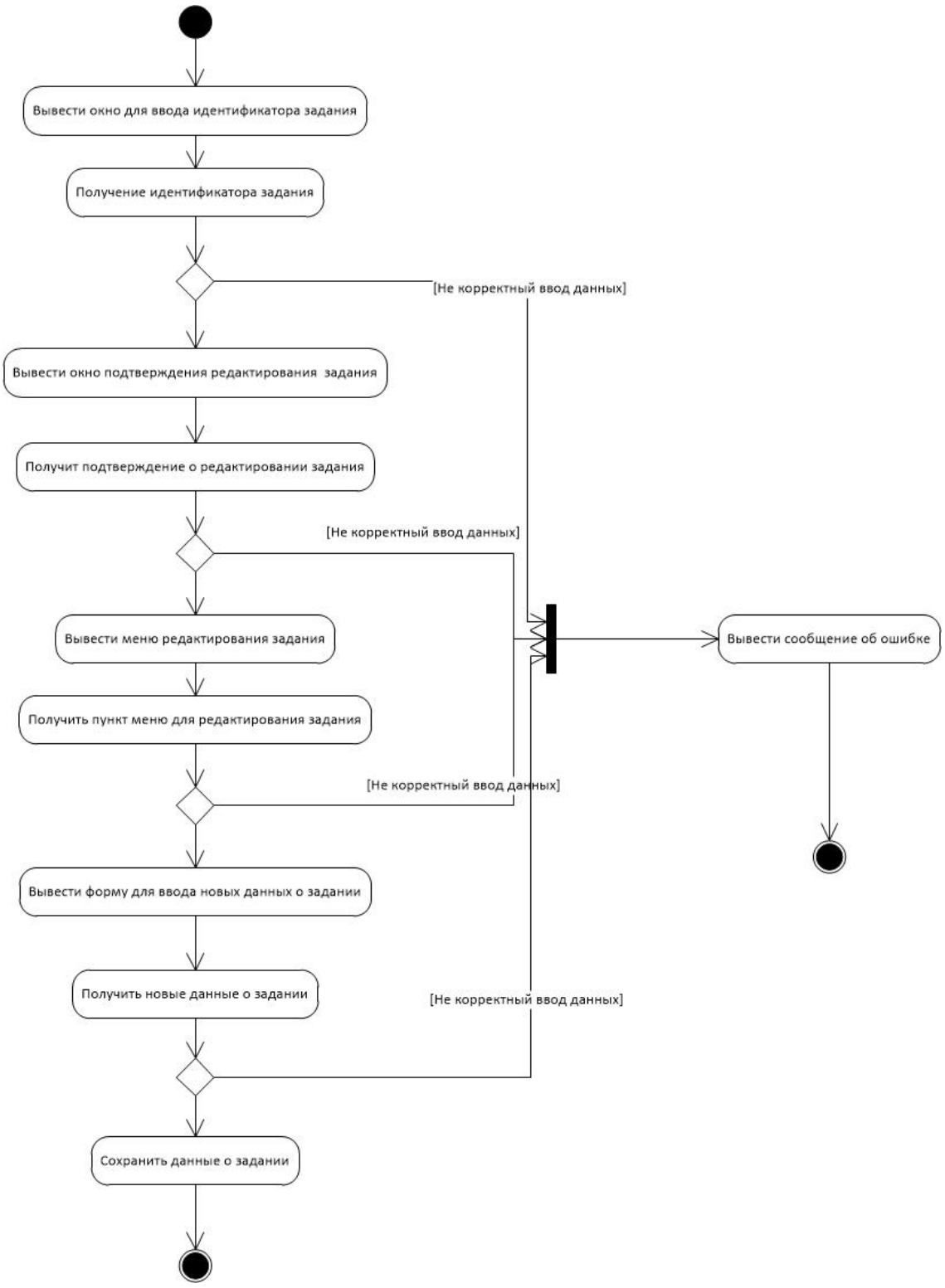

Рисунок 2.11 – Редактирование задания

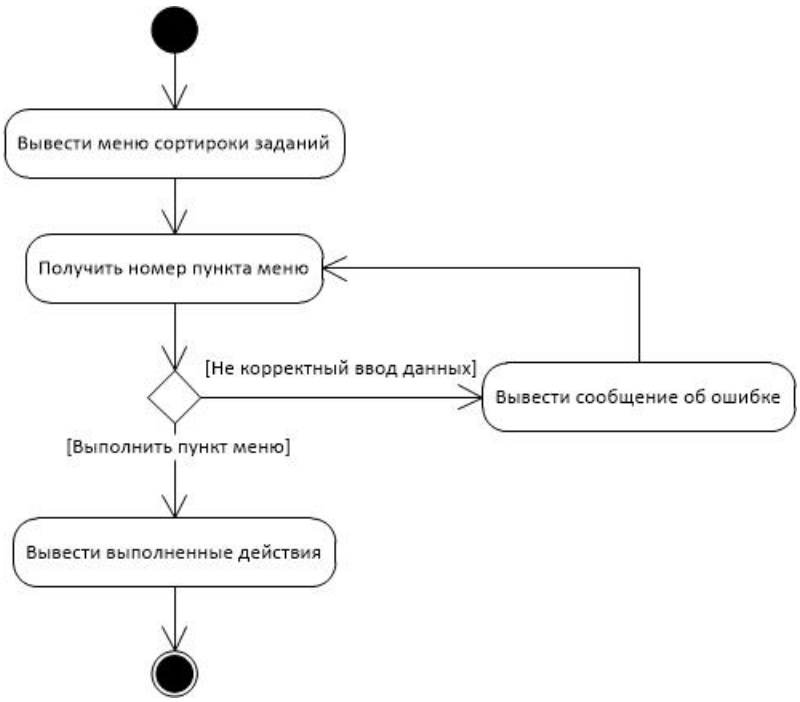

Рисунок 2.12 – Меню сортировки заданий

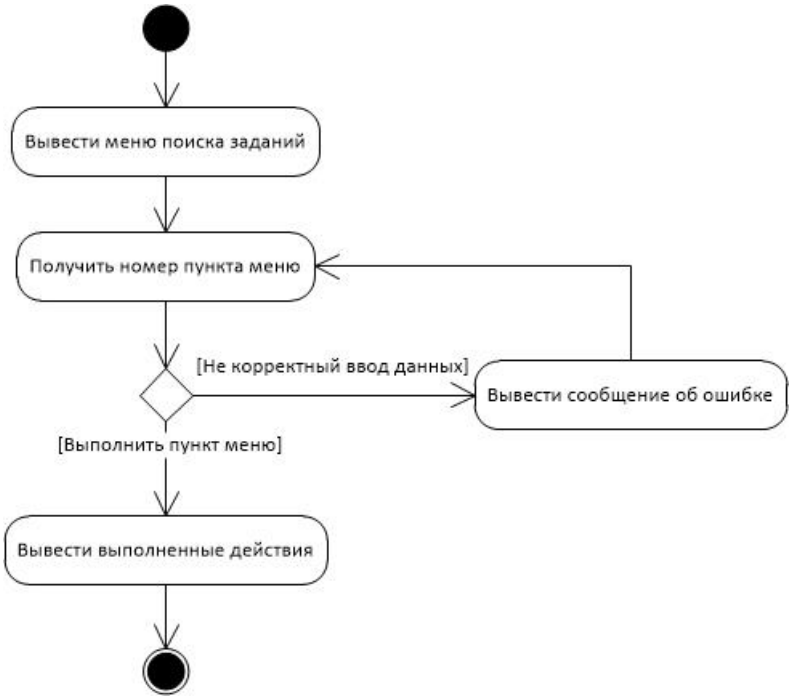

Рисунок 2.12 – Меню поиска заданий

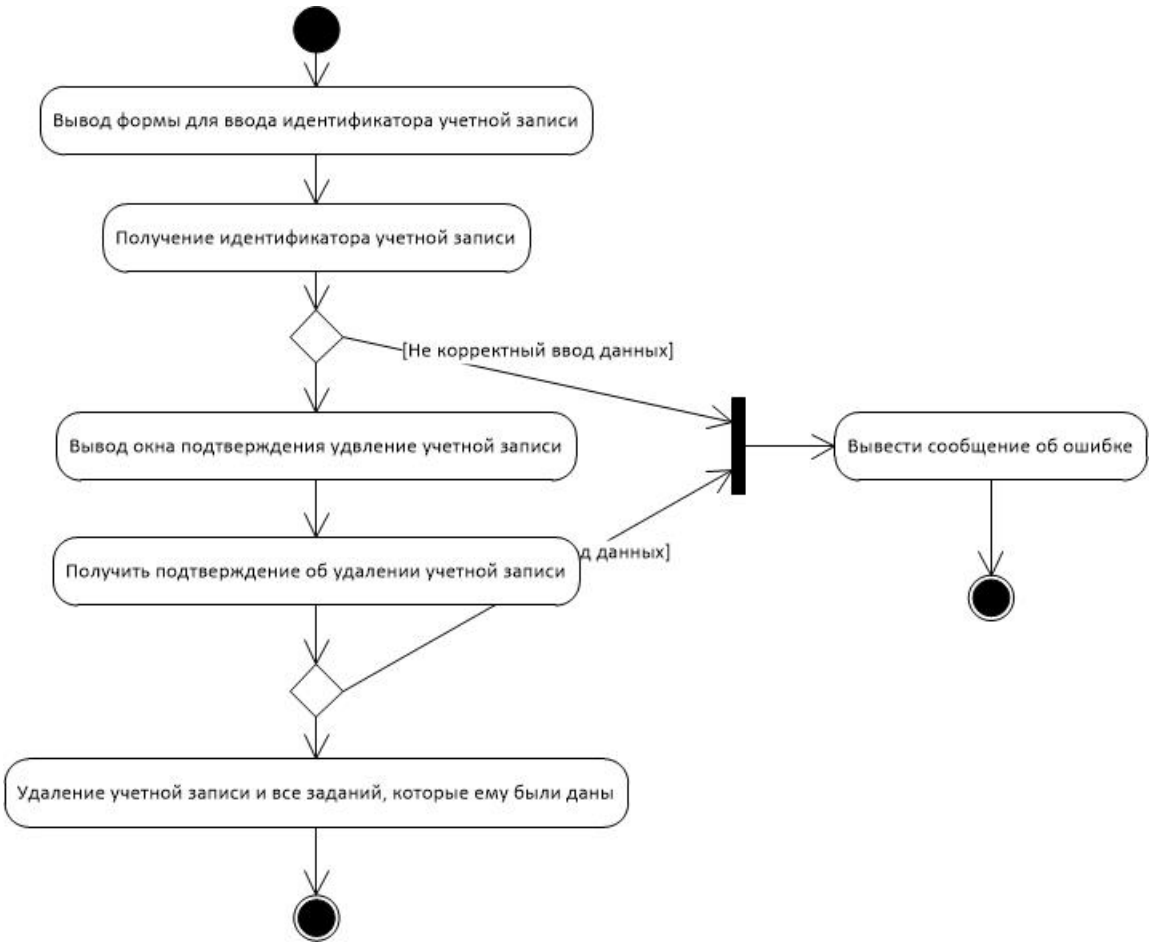

Рисунок 2.13 – Удаление учетной записи и его заданий

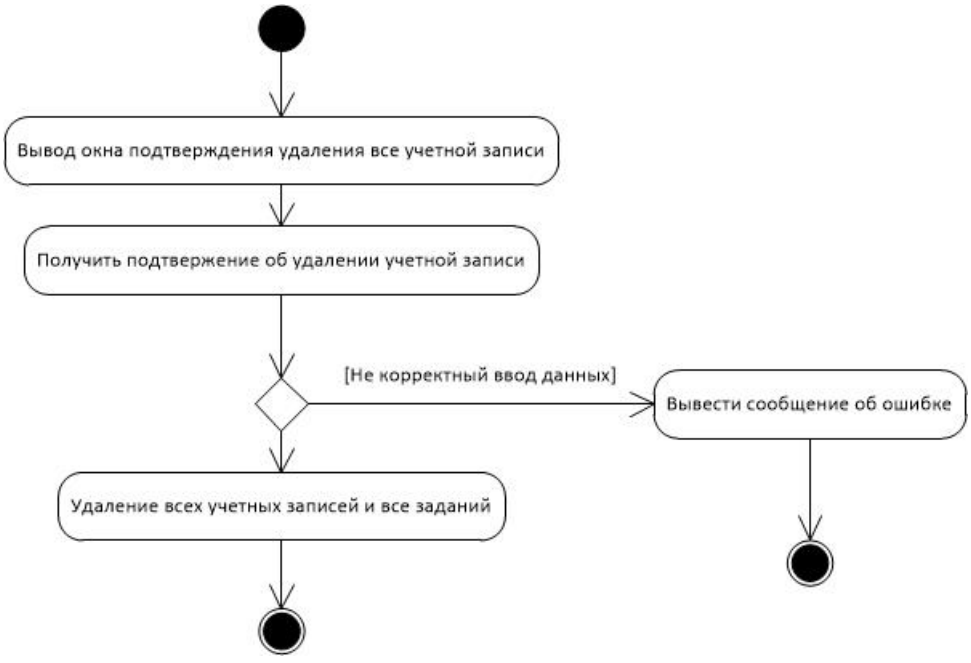

Рисунок 2.14 – Удаление всех учетных записей и заданий

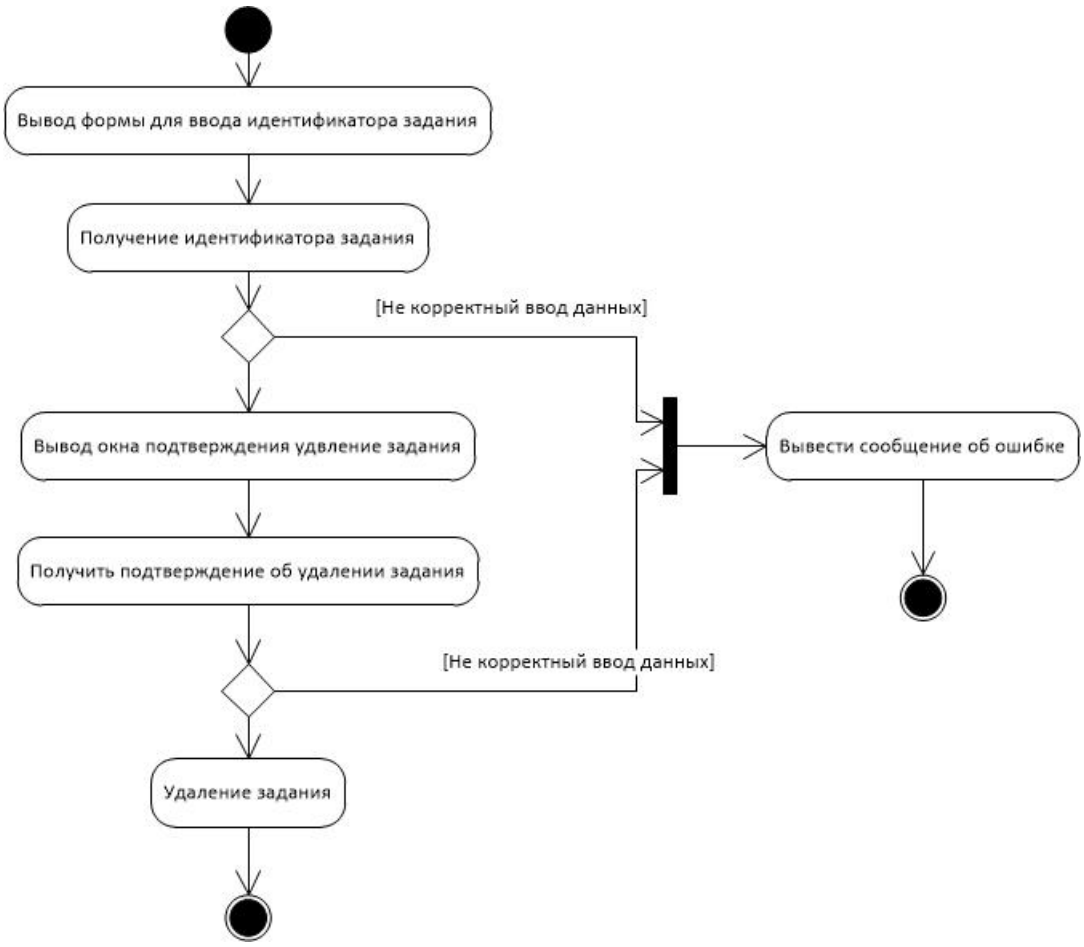

Рисунок 2.15 – Удаление задания

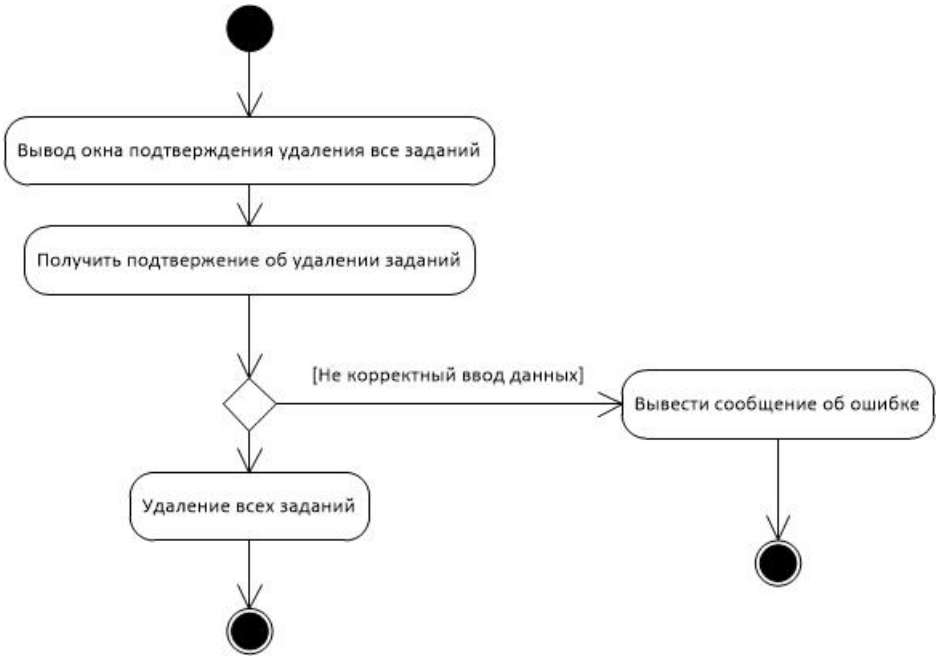

Рисунок 2.16 – Удаление всех заданий

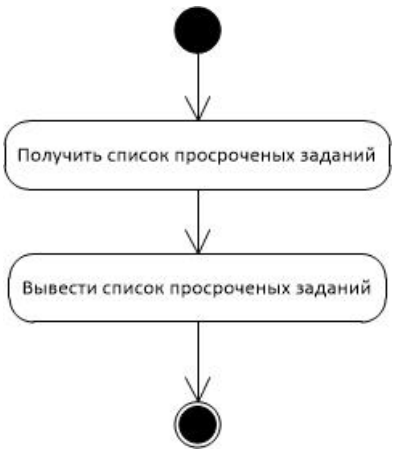

Рисунок 2.17 – Список просроченных заданий

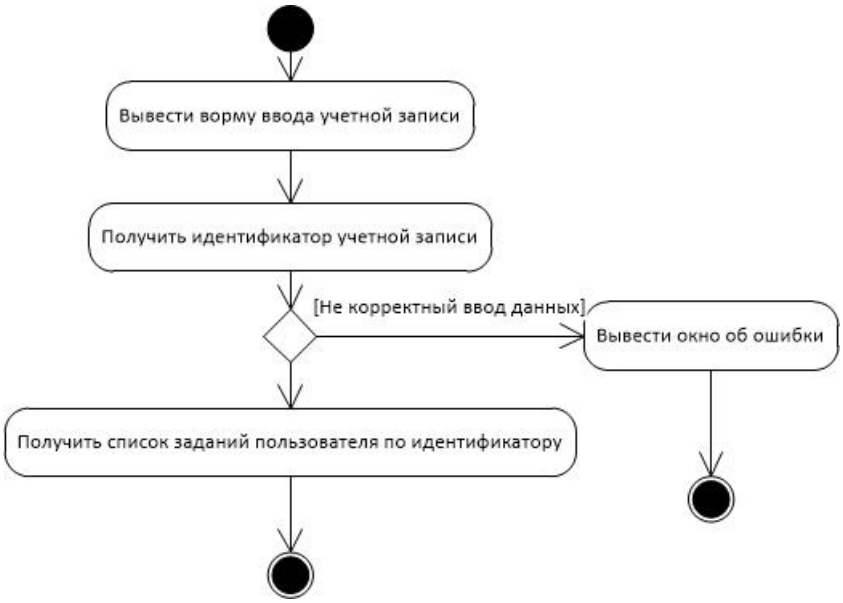

Рисунок 2.18 – Список заданий определенного пользователя

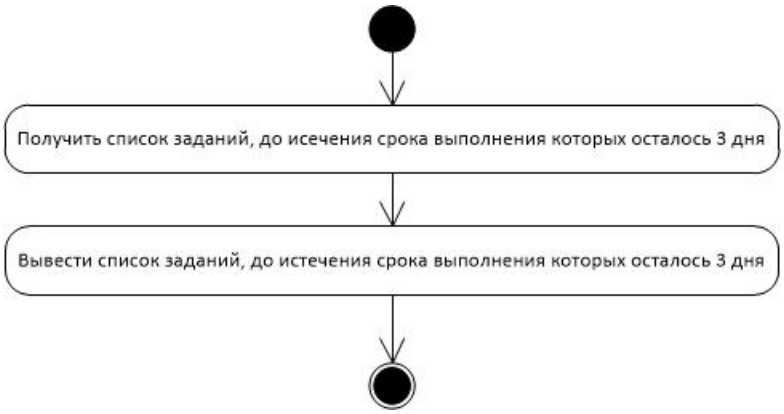

Рисунок 2.19 – Список заданий, до истечения срока сдачи которых осталось 3 дня.

3 ОПИСАНИЕ ПРОГРАММНОГО КОДА

3.1 Архитектура программы

В программе используются три класса для хранения данных об учетных записях пользователей (UserAccount), заданий для каждого из этих пользователей (Task) и класс для хранения даты (Date), которая используется в классе Task.

Класс ***UserAccount*** выполняет функцию хранения данных об учетной записи пользователя и выполнения операций ввода данных об учетной записи пользователя с консоли, вывод данных в виде анкеты, запись и считывание одной записи об учетной записи пользователя из файла, выполнение логических операций, позволяет авторизоваться и исходя из учетной записи, пользователь будет иметь доступ к определенным средствам работы с данными. Если пользователь вошел под аккаунтом администратора, то пользователь имеет полный доступ ко всем возможностям программы. Если пользователь вошел под другим аккаунтом, то пользователь имеет ограниченный доступ к возможностям программы.

Класс ***Task*** выполняет функцию хранения данных о задании и выполнения операций ввода данных о задании с консоли, вывод данных в виде анкеты, запись и считывание одной записи о задании из файла, выполнение логических операций.

Класс ***Date*** выполняет функцию хранения данных о дате задании и выполнения операций ввода данных о дате с консоли, вывод данных в форматированном виде, запись и считывание одной записи о задании из файла, выполнение логических и арифметических операций, дополнительно реализована функция получения сегодняшней даты.

В программе используются ряд классов и функций для обработки основных и промежуточных данных (краткое описание архитектуры программы):

– Класс *ServiceUserAccounts* выполняет функции сравнения, поиска, добавления, изменения, удаления учетных записей пользователей.

– Класс *ServiceTasks* выполняет функции сравнения, поиска, добавления, изменения, удаления заданий для сотрудников предприятия.

– Класс *LoginAndPasswordScreen* выполняет функцию вывода формы для входа в систему, осуществляет проверку введенных логина и пароля.

– Класс *AdministratorScreen* выполняет функцию интерфейса со всем возможностями программы.

– Класс *UserScreen* выполняет функцию интерфейса с ограниченными возможностями программы.

– Класс *HashPasswordWithSal* выполняет функцию шифрования пароля (в качестве соли используется фамилия учетной записи пользователя).

– Методы в head-файле *ExceptionScreen.h* выполняют функцию уведомления пользователя о различных ошибках.

– Методы в head-файле *ServiceData*.h выполняет функцию различных манипуляций над разными видами данных.

– Методы в head-файле *ControllerIOFStream.h* выполняют функцию чтения и запись в файлы различных видов данных.

– Методы в head-файле *ControllerIStream.h* выполняют функцию ввода вывода различных типов данных с клавиатуры.

– *main.cpp* запускает программу, используя выше перечисленные классы и методы, создает необходимые экземпляры классов, считывает данные из файлов, запрашивает логин и пароль и в зависимости от учетной записи выводит соответствующие интерфейсы.

3.2 Описание диаграммы классов

В диаграмме классов представлено 9 классов: Date, Task, UserAccount, ServiceTasks, ServiceUserAccounts, AdministratorScreen, UserScreen, LoginAndPasswordScreen, HashPasswordWithSal.

Диаграмма классов представлена в приложении Б.

Исходный код программы в приложении В.

Программа разработана в IDE Eclipse Cpp Oxygen.

3.2.1 Класс Date

Диаграмма классов представлена в приложении Б.

Класс Date имеет три **поля** (все приватные): day – день, month – месяц, year – год, тип данных которых short int.

Класс Date имеет три **конструктора** (все публичные): конструктор по умолчанию, конструктор с параметрами и конструктор копии.

Класс Date имеет **9 методов** (все публичные):

– 3 set-метода, которые соответствуют полям класса;

– 3 get-метода, которые соответствуют полям класса;

– Метод string toString() – из полей создает информационную строку с информацией об экземпляре класса;

– Метод int comparisonDate(const Date& obj) в качестве параметра получает ссылку на сравниваемый экземпляра класса Date, сравнивает полученный объект с объектом вызывающий этот метод. Возвращает -1 если принятый объект меньше вызываемого, 0 если равны, 1 если принятый объект больше вызываемого.

– Статический метод Date getTodayDate() – возвращает сегодняшнюю дату.

Класс Date имеет **8 операторов** (все публичные):

– Оператор присваивания (=);

– Оператор меньше (<);

– Оператор сравнения (==);

– Оператор минус (-) – позволяет получить разность дней между датами;

– Оператор ввода (>>) – позволяет вводить информацию об экземпляре класса;

– Оператор вывода (<<) – позволяет выводить информацию об экземпляре класса;

– Перегруженный оператор ввода (>>) – позволяет считывать информацию из файла;

– Перегруженный оператор вывода (<<) – позволяет записывать информацию в файл;

3.2.2 Класс Task

Класс Task имеет **8 полей** (все публичные кроме последнего):

– taskId – идентификатор задания, тип которого unsigned int;

– content – задание, тип которого string;

– userId – идентификатор пользователя, которому принадлежит задание, тип которого unsigned int;

– assignmentDate – дата выдачи задания, тип которого Date;

– plannedDate – планируемая дата сдачи задания, тип которого Date;

– dateDelivery – дата сдачи задания, тип котороgoDate;

– notation – , тип которого string;

– PATH_TASKS – хранит путь к файлу, где находятся все данные о заданиях, является статическим константным полем, тип которого string.

Класс Task имеет **3 конструктора** (все публичные): конструктор по умолчанию, конструктор с параметрами и конструктор копии.

Класс Task имеет **15 методов** (все публичные):

– 7 set-методы, которые соответствуют каждому полю кроме PATH_TASKS

– 7 get-методы, которые соответствуют каждому полю кроме PATH_TASKS

– Метод string toString() – из полей создает информационную строку с информацией об экземпляре класса;

Класс Task имеет **7 операторов** (все публичные):

– Оператор присваивания (=);

– Оператор меньше (<);

– Оператор сравнения (==);

– Оператор ввода (>>) – позволяет вводить информацию об экземпляре класса;

– Оператор вывода (<<) – позволяет выводить информацию об экземпляре класса;

– Перегруженный оператор ввода (>>) – позволяет считывать информацию из файла;

– Перегруженный оператор вывода (<<) – позволяет записывать информацию в файл;

3.2.3 Класс UserAccount

Класс UserAccount имеет **7 полей** (все приватные кроме последнего):

– userId – идентификатор учетной записи пользователя, тип которого unsigned int;

– password – зашифрованный пароль, тип которого string;

– surname – фамилия, тип которого string;

– name – имя, тип которого string;

– midname – отчество, тип которого string;

– department – отдел, тип которого string;

– PATH_USER_ACCOUNTS – хранит путь к файлу, где находятся все данные об учетных записях пользователя, является статических константным полем, тип которого string.

Класс UserAccount имеет **3 конструктора** (все публичные): конструктор по умолчанию, конструктор с параметрами и конструктор копии.

Класс UserAccount имеет **13 методов** (все публичные):

– 8 set-методы, которые соответствуют каждому полю кроме PATH_USER_ACCOUNTS

– 8 get-методы, которые соответствуют каждому полю кроме PATH_USER_ACCOUNTS

– Метод string toString() – из полей создает информационную строку с информацией об экземпляре класса;

Класс UserAccount имеет **7 операторов** (все публичные):

– Оператор присваивания (=);

– Оператор меньше (<);

– Оператор сравнения (==);

– Оператор ввода (>>) – позволяет вводить информацию об экземпляре класса;

– Оператор вывода (<<) – позволяет выводить информацию об экземпляре класса;

– Перегруженный оператор ввода (>>) – позволяет считывать информацию из файла;

– Перегруженный оператор вывода (<<) – позволяет записывать информацию в файл;

3.2.4 Класс ServiceTasks

Класс ServiceTasks имеет **одно публичное поле** PATH_FREE_TASKID – хранит путь к файлу, где хранятся освободившиеся идентификаторы для заданий, является статическим и константным, тип которого string.

Класс ServiceTasks имеет **3 конструктора** (все приватные): конструктор по умолчанию, конструктор с параметрами и конструктор копии.

Класс ServiceTasks имеет один приватный оператор присваивания (=).

Так как конструкторы и оператор присваивания приватные, то невозможно создать экземпляр этого класса ServiceTasks.

Класс ServiceTasks имеет **33 метода** (все публичные и статические):

1) Методы для редактирования данных заданий:

– Метод void addTask(list<Task> &ListTasks, list<UserAccount> &ListUserAccounts, list<unsigned int> &ListFreeTaskId) – в качестве входных параметров получает ссылку на список заданий (list<Task> &ListTasks) для создания и добавления нового задания, ссылку на список сотрудников (list<UserAccount> &ListUserAccounts) предприятия для присвоения задания конкретному сотруднику, ссылку на список свободных идентификаторов (list<unsigned int> &ListFreeTaskId) для присвоения заданию свободного идентификатора, если этот список пуст, то выбирается максимально большой идентификатор из списка заданий, инкрементируется и присваивается новому заданию. Метод требует от пользователя ввести данные о задании с клавиатуры.

Внимание! При этом идет проверка на:

– введенные дату выдачи задания и дату планируемой сдачи задания. Если дата выдачи задания больше даты планируемой сдачи, то отобразится соответствующую ошибку на экран пользователя.

– введение корректного идентификатора пользователя. Если идентификатор не соответствует ни одному из существующих пользователей, то отображается соответствующая ошибка на экран пользователя.

– Метод void editingContentTask(list<Task>::iterator &findTask) – в качестве входного параметра получает итератор на задание, запрашивает от пользователя текстовое задание, пере инициализирует соответствующее поле.

– Метод void editingUserIdTask(list<Task>::iterator &findTask, list<UserAccount> &ListUserAccounts) – в качестве входных параметров получает итератор на задание и список сотрудников предприятия, запрашивает от пользователя идентификатор учетной записи которому будет присвоено это задание, пере инициализирует соответствующее поле.

Внимание! При этом идет проверка на введение корректного идентификатора пользователя. Если идентификатор не соответствует ни одному из существующих пользователей, то отображается соответствующая ошибка на экран пользователя.

– Метод void editingPlannedDateTask(list<Task>::iterator &findTask) – в качестве входного параметра получает итератор на задание, запрашивает от пользователя планируемую дату сдачи задания, пере инициализирует соответствующее поле.

Внимание! При этом идет проверка на введенную дату планируемой сдачи задания и сравнивает ее с датой выдачи задания. Если дата выдачи задания больше даты планируемой сдачи, то отобразится соответствующая ошибка на экран пользователя.

– Метод void editingDateDeliveryTask (list<UserAccount>::iterator &activeAccount, list<Task>::iterator &findTask) – в качестве входных параметров получает итератор на пользователя, который вошел в программу, и итератор на задание, пере инициализирует соответствующее поле сегодняшней датой.

Внимание! При этом идет проверка на то кто вошел в систему. Если в программу вошли под учетной записью администратора и дата сдачи задания просрочена, то дата инициализируется. Если в программу вошли под другим аккаунтом и дата сдачи задания просрочена, то выводиться сообщение о том, что пользователь не имеет право редактировать дату сдачи задания и должен обратиться к администратору.

– Метод void editingNotationTask(list<Task>::iterator &findTask) – в качестве входного параметра получает итератор на задание, запрашивает от пользователя текстовое примечание, пере инициализирует соответствующее поле.

– Метод void editingAllDataTask(list<Task>::iterator &findTask, list<UserAccount> &ListUserAccounts) – в качестве входных параметров получает итератор на задание и список учетных записей пользователей, запрашивает от пользователя текстовое, идентификатор пользователя, которому будет присвоено это задание, планируемую дату сдачи, текстовое примечание, пере инициализирует все введенные данные соответствующих полей с исправлением даты выдачи задания на сегодняшнюю дату.

Внимание! При этом идет проверка на введение корректного идентификатора пользователя. Если идентификатор не соответствует ни одному из существующих пользователей, то отображается соответствующая ошибка на экран пользователя. Если идентификатор соответствует одному из существующих пользователей, то идет проверка на введенную планируемую дату сдачи задания. Если дата выдачи задания больше даты планируемой сдачи, то отобразится соответствующая ошибка на экран пользователя.

– Метод void deleteTask(list<Task> &ListTasks, list<unsigned int> &ListFreeTaskId) – в качестве входных параметров получает список заданий и список свободных идентификаторов для заданий, запрашивает от пользователя идентификатор задания, данные которого необходимо удалить, если задание с введенным идентификатором существует, то программа запрашивает подтверждение удаления задания, происходит удаление.

Внимание! При этом идет проверка на введение корректного идентификатора задания. Если не существует задания с таким идентификатором, то программа отображает соответствующую ошибку. Если существует задания с таким идентификатором, то программа проверяет корректность ввода подтверждения. Если введены некорректные данные подтверждения, то выводит соответствующую ошибку.

– Метод void deleteAllTask(list<Task> &ListTasks, list<unsigned int> &ListFreeTaskId) – в качестве входных параметров получает ссылки на список заданий и список свободных идентификаторов для заданий,

запрашивает подтверждение удаления задания, происходит удаление всех заданий.

Внимание! При этом идет проверка корректности ввода подтверждения. Если введены некорректные данные подтверждения, то выводит соответствующую ошибку.

– Метод void ServiceTasks::deleteTaskByIterator(list<Task> &ListTasks, list<Task>::iterator &findTask, list<unsigned int> &ListFreeTaskId) – в качестве входных параметров принимает ссылки на список заданий, итератор на задание, список свободных идентификаторов для заданий, добавляет в список свободных идентификаторов для заданий идентификаторов, который принадлежал удаленному заданию, задание удаляется из список заданий;

2) Методы для обработки данных заданий (не редактируемые методы):

– Методы компараторы

```
bool compTaskByTaskId(Task& first, Task& second);
bool compTaskByContent(Task& first, Task& second);
bool compTaskByAssignmentDate(Task& first, Task& second);
bool compTaskByPlannedDate(Task& first, Task& second);
bool compTaskByDateDelivery(Task& first, Task& second);
bool compTaskByNotation(Task& first, Task& second);
bool recompTaskByTaskId(Task& first, Task& second);
bool recompTaskByContent(Task& first, Task& second);
bool recompTaskByUserId(Task& first, Task& second);
bool recompTaskByAssignmentDate(Task& first, Task& second);
bool recompTaskByPlannedDate(Task& first, Task& second);
bool recompTaskByDateDelivery(Task& first, Task& second);
bool recompTaskByNotation(Task& first, Task& second);
```

Служат для применения в сортировке списка заданий, выполняют функцию сравнения двух экземпляров класса Task и возвращают значение логического тепа данных. Методы с приставкой re- возвращают инверсное значение логического типа данных. Каждый метод соответствует одному из полей класса Task.

– Методы поиска

```
list<Task>::iterator findTaskByTaskId( list<Task>::iterator begin,
list<Task>::iterator end, unsigned int taskId);
list<Task>::iterator findTaskByContent( list<Task>::iterator begin,
list<Task>::iterator end, string content);
list<Task>::iterator findTaskByUserId(list<Task>::iterator begin,
list<Task>::iterator end, unsigned int userId);
list<Task>::iterator findTaskByAssignmentDate(list<Task>::iterator begin,
list<Task>::iterator end, Date assignmentDate);
list<Task>::iterator findTaskByPlannedDate(list<Task>::iterator begin,
list<Task>::iterator end, Date plannedDate);
list<Task>::iterator findTaskByDateDelivery(list<Task>::iterator begin,
list<Task>::iterator end, Date dateDelivery);
```

list<Task>::iterator findTaskByNotation(list<Task>::iterator begin,
list<Task>::iterator end, string notation);

Выполняют функцию поиска экземпляра класса Task в соответствии с переданным параметром. В качестве входных параметров принимает пару итераторов на часть или весь список заданий и значение по которому будет производиться поиск. Каждый метод соответствует одному из полей класса Task. Метод находит первую встречающуюся запись и возвращает итератор на это задание. Если в списке отсутствует данная запись, то возвращает итератор на конец списка.

3) Методы для отображения статистики заданий (не редактируемые методы):

– Метод void showOverdueTasks(list<Task> &ListTasks) – в качестве входного параметра получает ссылку на список заданий, выводит на экран пользователя весь список просроченные задания. Если список пуст, то программа отображает соответствующее сообщение. Если таковых нет, то программа отображает соответствующее сообщение.

– Метод void showTasksByUserId(list<Task> &ListTasks) – в качестве входного параметра получает ссылку на список заданий, запрашивает пользователя ввести идентификатор пользователя, задания которого необходимо просмотреть, выводит список заданий конкретного пользователя. Если список пуст, то программа отображает соответствующее сообщение. Если таковых нет, то программа отображает соответствующее сообщение.

– Метод void showOverdueTasksToThreeDays(list<Task> &ListTasks) – в качестве входного параметра получает ссылку на список заданий, выводит список всех заданий до окончания сдачи которого осталось 3 дня. Если список пуст, то программа отображает соответствующее сообщение. Если таковых нет, то программа отображает соответствующее сообщение.

3.2.5 Класс ServiceUserAccounts

Класс ServiceUserAccounts имеет одно публичный поле PATH_FREE_USERID – хранит путь к файлу со списком свободных идентификаторов для учетных записей пользователей.

Класс ServiceUserAccounts имеет **3 конструктора** (все приватные): конструктор по умолчанию, конструктор с параметрами и конструктор копии.

Класс ServiceUserAccounts имеет один приватный оператор присваивания (=).

Так как конструкторы и оператор присваивания приватные, то невозможно создать экземпляр этого класса ServiceUserAccounts.

Класс ServiceUserAccounts имеет **30 методов** (все публичные и статические):

1) Методы для редактирования данных учетных записей пользователей:

– Метод void addUserAccount(list<UserAccount> &ListUserAccounts, list<unsigned int> &ListFreeUserId) – в качестве входных параметров получаем ссылку на список учетных записей пользователей и на список свободных идентификаторов учетных записей пользователей для присвоения учетной записи пользователя свободного идентификатора, если этот список пуст, то выбирается максимально большой идентификатор из списка учетных записей пользователей, инкрементируется и присваивается новой учетной записи пользователя. Метод требует от пользователя ввести данные об учетной записи пользователя с клавиатуры. После программа требует подтверждение о создании новой учетной записи пользователя пользователем. Новый пользователь успешно создан и добавлен в список.

Внимание! При этом идет проверка на корректное подтверждение. Если введенное подтверждение не корректно, то на экране пользователя выводиться соответствующее сообщение.

– Метод void editingSNMUserAccount(list<UserAccount> ::iterator& it, string passwordAccount) – в качестве входных параметров получает ссылку на итератор на учетную запись пользователя и пароль этого аккаунта. Программа требует ввести фамилию, имя, отчество с клавиатуры. Метод перекодирует пароль, так как в качестве соли используется фамилия учетной записи пользователя. Пере инициализирует соответствующие поля.

– Метод void editingDepartmentUserAccount (list<UserAccount> ::iterator& it) – в качестве входного параметра получает ссылку на итератор учетной записи пользователя. Программа требует ввести отдел. Пере инициализация соответствующего поля.

– Метод void editingPasswordUserAccount(list<UserAccount>::iterator& it, string passwordAccount) – в качестве входных параметров получает ссылку на итератор на учетную запись пользователя и пароль этого аккаунта. Программа требует ввести новый пароль. Метод кодирует новый пароль. Пере инициализация соответствующего поля.

– Метод void editingAllDataUserAccount (list<UserAccount>::iterator& it) – в качестве входного параметра получает ссылку на итератор учетной записи пользователя. Метод требует ввести пароль, фамилию, имя, отчество, отдел. Пере инициализация соответствующего поля.

– Метод void deleteUserAccountAndTasks(list<UserAccount> &ListUserAccounts, list<Task> &ListTasks, list<unsigned int> &ListFreeUserId, list<unsigned int> &ListFreeTaskId) – в качестве входных параметров получает ссылку на список учетных записей пользователей, список заданий, список свободных идентификаторов учетных записей пользователей, список свободных идентификаторов заданий, требует от пользователя ввести

идентификатор учетной записи пользователя которую необходимо удалить, после требует подтверждение об удалении, удаляет.

Внимание! При этом идет проверка на:

– Если в качестве идентификатора учетной записи пользователя введен идентификатор администратора, то метод отображает, что администратора удалить нельзя.

– Если учетная запись пользователя по идентификатору не найдена, то высвечивается соответствующее сообщение.

– Если некорректно введены данные по подтверждению, то отображается соответствующее сообщение.

– Метод void ServiceUserAccounts::deleteAllUserAccountsAndTasks(list< UserAccount> &ListUserAccounts, list<Task> &ListTasks, list<unsigned int> &ListFreeUserId, list<unsigned int> &ListFreeTaskId) – в качестве входных параметров получает ссылку на список учетных записей пользователей, список заданий, список свободных идентификаторов учетных записей пользователей, список свободных идентификаторов заданий, требует подтверждение об удалении, удаляет.

Внимание! При этом идет проверка на:

– Если учетная запись пользователя по идентификатору не найдена, то высвечивается соответствующее сообщение.

– Если некорректно введены данные по подтверждению, то отображается соответствующее сообщение.

– Метод void deleteUserAccountAndTasksByIterator (list <UserAccount> &ListUserAccounts, list<Task> &ListTasks, list<UserAccount>::iterator &findUserAccount, list<unsigned int> &ListFreeUserId, list<unsigned int> &ListFreeTaskId) – в качестве входных параметров получает ссылку на список учетных записей пользователей, список заданий, итератор на учетную запись пользователя, список свободных идентификаторов учетных записей пользователей, список свободных идентификаторов заданий, метод удаляет все задания, которые принадлежат этому пользователю, и добавляет их идентификаторы в список свободных идентификаторов заданий, после удаляет учетную запись пользователя и добавляет добавляет их идентификаторы в список свободных идентификаторов учетных записей пользователей.

2) Методы для обработки данных учетных записей пользователей (не редактируемые):

– Методы компараторы

```
bool compUserAccountByUserId(UserAccount& first, UserAccount& second);
bool compUserAccountByName(UserAccount& first, UserAccount& second);
bool compUserAccountByMidname(UserAccount& first, UserAccount& second);
bool compUserAccountByDepartment(UserAccount& first, UserAccount&
second);
bool recompUserAccountByUserId(UserAccount& first, UserAccount& second);
```

bool recompUserAccountBySurname(UserAccount& first, UserAccount& second);
bool recompUserAccountByName(UserAccount& first, UserAccount& second);
bool recompUserAccountByMidname(UserAccount& first, UserAccount& second);
bool recompUserAccountByDepartment(UserAccount& first, UserAccount & second);

Служат для применения в сортировке списка учетных записей пользователей, выполняют функцию сравнения двух экземпляров класса UserAccount и возвращают значение логического тепа данных. Методы с приставкой re- возвращают инверсное значение логического типа данных. Каждый метод соответствует одному из полей класса UserAccount.

– Методы поиска

list<UserAccount>::iterator findUserAccountByUserId(list<UserAccount> ::iterator begin, list<UserAccount>::iterator end, unsigned int userId);
list<UserAccount>::iterator findUserAccountByPassword(list<UserAccount> ::iterator begin, list<UserAccount>::iterator end, string password);
list<UserAccount>::iterator findUserAccountBySurname(list<UserAccount> ::iterator begin, list<UserAccount>::iterator end, string surname);
list<UserAccount>::iterator findUserAccountByName(list<UserAccount> ::iterator begin, list<UserAccount>::iterator end, string name);
list<UserAccount>::iterator findUserAccountByMidname(list<UserAccount> ::iterator begin, list<UserAccount>::iterator end, string midname);
list<UserAccount>::iterator findUserAccountByDepartment(list<User Account> ::iterator begin, list<UserAccount>::iterator end, string department);

Выполняют функцию поиска экземпляра класса UserAccount в соответствии с переданным параметром. В качестве входных параметров принимает пару итераторов на часть или весь список учетных записей пользователей и значение по которому будет производиться поиск. Каждый метод соответствует одному из полей класса UserAccount. Метод находит первую встречающуюся запись и возвращает итератор. Если в списке отсутствует данная запись, то возвращает итератор на конец списка.

3.2.6 Класс LoginAndPasswordScreen

Класс LoginAndPasswordScreen имеет **2 поля** (все приватные):

– ListUserAccounts – список учетных записей пользователей, являеться указателем, тип которого list<UserAccount>.

– created – метка, которая показывает о существовании объекта, являеться статической переменной, тип которой bool.

Класс LoginAndPasswordScreen имеет **2 конструктор** (все приватные): конструктор параметров, конструктор копии.

Класс LoginAndPasswordScreen имеет **один приватный оператор** присваивания.

Так как конструкторы и оператор присваивания приватные, то невозможно создать экземпляр этого класса LoginAndPasswordScreen.

Класс LoginAndPasswordScreen имеет **3 метода**:

– Статический метод LoginAndPasswordScreen& instance(list<UserAccount> &ListUserAccounts) – в качестве входного параметра получает ссылку на список учетных записей пользователей, метод вызывает приватный конструктор LoginAndPasswordScreen только один раз за всю работу программы и возвращает ссылку на созданный объект класса LoginAndPasswordScreen.

– Метод list<UserAccount>::iterator showLoginAndPasswordScreen() – используеться попарно со статическим методом, указанный выше. Запрашивает логин и пароль и дает доступ к программным средствам в зависимости от аккаунта, под которым вошли в систему.

– Приватный метод list<UserAccount>::iterator inputAndCheckLoginAndPassword – вызывается в метода описанном выше для ввода логина и пароля, возвращает итератор на аккаунт под которым вошли в систему.

Внимание! При этом идет проверка на:

– Если учетная запись пользователя по идентификатору не найдена, то высвечивается соответствующее сообщение.

– Если некорректно введен пароль, то отображается соответствующее сообщение.

3.2.7 Класс AdministratorScreen

Класс AdministratorScreen имеет **5 полей** (все приватные):

– ListUserAccounts – указатель на список учетных записей пользователей, тип которого list<UserAccount>.

– ListTasks – указатель на список заданий, тип которого list<Task>.

– ListFreeUserId – список свободных идентификаторов учетных записей пользователей, тип которого list<unsigned int>.

– ListFreeTaskId – список свободных идентификаторов заданий, тип которого list< unsigned int>.

– created – метка о существовании экземпляра класса AdministratorScreen.

Класс AdministratorScreen имеет **2 конструктора** (все приватные): конструктор параметров, конструктор копии.

Класс AdministratorScreen имеет **один приватный оператор** присваивания.

Так как конструкторы и оператор присваивания приватные, то невозможно создать экземпляр этого класса AdministratorScreen.

Класс AdministratorScreen имеет **12 методов**:

– Статический метод AdministratorScreen& instance(list<UserAccount> &ListUser Accounts, list<Task> &ListTasks) – в качестве входных параметров

передается ссылка на список учетных записей пользователей и список заданий. Метод вызывает приватный конструктор класса AdministratorScreen и не позволяет создавать больше чем один экземпляр этого класса. Особенность этого конструктора в том, что он считывает из файлов данные о свободных идентификаторов учетных записей пользователей и свободных идентификаторов заданий, пере инициализирует соответствующие поля. Возвращает ссылку на экземпляр этого класса.

– Метод void showAdministratorScreen() – используется попарно со статическим методам описанном выше. Отображает главное меню для администратора. Позволяет переходить к другим пунктам меню, позволяет сохранять изменения в соответствующие файлы и прекратить работу программы.

– Приватный метод void accountManagementMenu() – отображает меню управления учетными записями пользователя.

– Приватный метод void tasksAndStatisticsMenu() – отображение меню управления заданиями.

– Приватный метод void editingUserAccountMenu() – отображает меню редактирования учетных записей пользователей.

– Приватный метод void editingTaskMenu() – отображает меню редактирования заданий.

– Приватный метод void sortInAscendingOrderUserAccountsMenu() – отображает меню сортировки списка учетных записей пользователей по возрастанию.

– Приватный метод void sortInAscendingOrderTasksMenu() – отображает меню сортировки списка заданий по возрастанию.

– Приватный метод void sortInDecreasingOrderUserAccountsMenu() – отображает меню сортировки списка учетных записей пользователей по убыванию.

– Приватный метод void sortInDecreasingOrderTasksMenu() – отображает меню сортировки списка заданий по убыванию.

– Приватный метод void findUserAccountMenu() - отображает меню поиска учетных записей пользователей.

– Приватный метод void findTaskMenu() - отображает меню поиска заданий.

3.2.8 Класс UserScreen

Класс UserScreen имеет **4 поля** (все приватные):

– ListTasks – указатель на список заданий, тип которого list<Task>.

– ListTasksBuff – список на буферный список заданий, тип которого list<Task>.

– activeAccount – ссылка на итератор на учетную запись пользователя, под которым вошли в систему, имеет тип list<UserAccount>::iterator.

– created – метка указывающая на существование экземпляра класса UserScreen.

Класс UserScreen имеет **2 конструктора** (все приватные): конструктор параметров, конструктор копии.

Класс UserScreen имеет **один приватный оператор** присваивания.

Так как конструкторы и оператор присваивания приватные, то невозможно создать экземпляр этого класса UserScreen.

Класс UserScreen имеет **5 методов**:

– Статический метод UserScreen& instance(list<Task> &ListTasks, list<User Account>::iterator &activeAccount) – в качестве входных параметров передается ссылка на список заданий и итератор на учетную запись пользоваться, под которым вошли в систему. Метод отбирает задания, которые принадлежат пользователю, под которым вошли в систему и хранит их в ListTasksBuff. Метод вызывает приватный конструктор класса UserScreen и не позволяет создавать больше чем один экземпляр этого класса. Возвращает ссылку на экземпляр этого класса.

– Метод void showUserScreen() – используется попарно со статическим методам описанном выше. Отображает главное меню для пользователя. Позволяет переходить к другим пунктам меню, позволяет сохранять изменения в соответствующие файлы и прекратить работу программы.

– Метод void sortInAscendingOrderTasksMenu() – отображает меню сортировки списка заданий по возрастанию.

– Приватный метод void sortInDecreasingOrderTasksMenu() – отображает меню сортировки списка заданий по убыванию.

– Приватный метод void findTaskMenu() - отображает меню поиска заданий.

3.2.9 Класс HashPasswordWithSal

Класс HashPasswordWithSal имеет только один публичный оператор (), который в качестве параметров пароль и соль (фамилию), кодирует пароль и возвращает закодированный пароль.

3.3 Описание функций, которые не вошли в диаграмму классов

Такими функциями являются main(), showExceptionScreen(), showNotData(), showNotFindData(), showNotFindUserId(), showNotCompDate(), showNotPlannedDate(), showList(), readFile(), writeFile(), inputString(), inputLine(), inputUnsignedInt(), inputShortInt().

3.3.1 Описание функций из cpp-файла main.cpp

В файле main.cpp определена только одна функция main() которая задает кодировку консоли 1251, создает и считывает списки учетных записей

пользователей и заданий, отображает экран ввода логина и пароля, после выводит соответствующий интерфейс для определенной учетной записи пользователя, под которой вошли в систему.

3.3.2 Описание функций из head-файл ExceptionScreen.h

В head-файле ExceptionScreen.h определены 6 функций для отображения на экран каких-либо ошибок при работе с программой:

– Функция showExceptionScreen() – вызывается при вводе не корректных данных, отображает сообщение вида «ОШИБКА ВВОДА ДАННЫХ! ПОПРОБУЙТЕ ЕЩЕ РАЗ!».

– Функция showNotData() – вызывается при условии, когда программа не нашла записей соответствующих запросу пользователя, отображает сообщение вида «ПО ВАШЕМУ ЗАПРОСУ ДАННЫХ НЕ СУЩЕСТВУЕТ! ПОПРОБУЙТЕ ЕЩЕ РАЗ!».

– Функция showNotFindData() – вызывается при условии, когда список данных пуст, отображает сообщение вида «ДАННЫХ НЕТ В БАЗЕ ДАННЫХ! НЕТ ВОЗМОЖНОСТИ МАНИПУЛЯЦИЙ!».

– Функция showNotFindUserId() – вызывается при условии, когда в списке отсутствует запись пользователя с введенным идентификатором, отображает сообщение вида «ПОЛЬЗОВАТЕЛЯ С ТАКИМ ID = # НЕТ В БАЗЕ ДАННЫХ!».

– Функция showNotCompDate() – вызывается при условии, когда по случайности планируемая дата сдачи задания получилась меньше даты выдачи задания, отображает сообщение вида «ПЛАНИРУЕМАЯ ДАТА СДАЧИ НЕ МОЖЕТ БЫТЬ МЕНЬШЕ ДАТЫ ВЫДАЧИ ЗАДАНИЯ».

– Функция showNotPlannedDate() – вызывается при условии, когда пользователь хочет изменить дату сдачи задания, но он просрочил ее сдачу, отображает сообщение вида «ВРЕМЯ СДАЧИ ЗАДАНИЯ ПРОСРОЧЕНО ПО ЭТОЙ ПРИЧИНЕ ОБРАТИТЕСЬ К АДМИНИСТРАТОРУ».

3.3.3 Описание функций из head-файл ServiceData.h

В head-файле ServiceData.h определена только одна функция showList() для вывода на экран списка любых видов данных, так является шаблонной функцией.

3.3.4 Описание функций из head-файл ControllerIOFStream.h

В head-файле ControllerIOFStream.h определены 2 функции readFile() и writeFile() для считывания и запись в файлов списка любых видов данных, так как эти функции шаблонные.

3.3.5 Описание функций из head-файл ControllerIStream.h

В head-файле ControllerIStream.h определены 4 функции для ввода с клавиатуры некоторых типов данных: inputString() – позволяет вводить строку до первого пробела, inputLine() – позволяет вводить строку любой сложности, inputUnsignedInt() – позволяет вводить без знаковые целочисленные цифры, inputShortInt() – позволяет вводить короткие целочисленные цифры.

4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Данное программное обеспечение тестируется методом функционального тестирования. Это самый простой и естественный метод тестирования.

Для обнаружения ошибок необходимо выполнить тестирование программы вручную.

Метод очень эффективен, но не применим для больших программ, программ с трудными расчетами и когда ошибка исходит из неправильного понимания технического задания. Данный метод часто использует в качестве компонента других методов устранения неисправностей.

Функциональные тесты основываются на функциях, выполняемых системой, и могут проводиться на всех уровнях тестирования. Как правило, эти функции описываются в требованиях, функциональных спецификациях или в виде случаев использования системы (use cases). Единственным недостатком этого вида тестирование является возможность упущения логических ошибок.

В ходе разработки программы было проведено функциональное тестирование, состоящее из 15 тест-кейсов, которые соответствуют 15 таблицам.

Таблица 1 – Тестирование начало программы

Исходные данные
0 0000
Ожидаемый результат
Вы вошли под учетной записью ID#: 0 Фамилия: Бабанин Имя: Дмитрий Отчество: Сергеевич Отдел: Начальник
Результат работы программы
Вы вошли под учетной записью ID#: 0 Фамилия: Бабанин Имя: Дмитрий Отчество: Сергеевич Отдел: Начальник Для продолжения нажмите любую клавишу . . .

Таблица 2 – Тестирование главного меню администратора

Исходные данные
1
Ожидаемый результат
ID#: 0 Фамилия: Бабанин

Имя: Дмитрий Отчество: Сергеевич Отдел: Начальник ----------Меню управления учетными записями пользователя---------- 1. Добавить учетную запись пользователя 2. Редактировать учетную запись пользователя 3. Удаление учетной записи пользователя 4. Удаление всех учетных записей пользователей 5. Сортировка по возрастанию/А-Я 6. Сортировка по убыванию/Я-А 7. Поиск 0. Вернуться >>
Результат работы программы
ID#: 0 Фамилия: Бабанин Имя: Дмитрий Отчество: Сергеевич Отдел: Начальник ----------Меню управления учетными записями пользователя---------- 1. Добавить учетную запись пользователя 2. Редактировать учетную запись пользователя 3. Удаление учетной записи пользователя 4. Удаление всех учетных записей пользователей 5. Сортировка по возрастанию/А-Я 6. Сортировка по убыванию/Я-А 7. Поиск 0. Вернуться >>

Таблица 3 – Тестирование меню управления учетными записями

Исходные данные
0
Ожидаемый результат
----------Главное меню администратора---------- 1. Учетные записи пользователей 2. Задания и статистика 3. Сохранение изменений 0. Выйти >>
Результат работы программы
----------Главное меню администратора---------- 1. Учетные записи пользователей 2. Задания и статистика 3. Сохранение изменений 0. Выйти >>

Таблица 4 – Тестирование добавления учетной записи пользователя

Исходные данные
1111 Почивалин Никита Николаевич Сотрудник да
Ожидаемый результат
ID#: 0 Фамилия: Бабанин Имя: Дмитрий Отчество: Сергеевич Отдел: Начальник ID#: 1 Фамилия: Почивалин Имя: Никита Отчество: Николаевич Отдел: Сотрудник ----------Меню управления учетными записями пользователя---------- 1. Добавить учетную запись пользователя 2. Редактировать учетную запись пользователя 3. Удаление учетной записи пользователя 4. Удаление всех учетных записей пользователей 5. Сортировка по возрастанию/А-Я 6. Сортировка по убыванию/Я-А 7. Поиск 0. Вернуться >>
Результат работы программы
ID#: 0 Фамилия: Бабанин Имя: Дмитрий Отчество: Сергеевич Отдел: Начальник ID#: 1 Фамилия: Почивалин Имя: Никита Отчество: Николаевич Отдел: Сотрудник ----------Меню управления учетными записями пользователя---------- 1. Добавить учетную запись пользователя 2. Редактировать учетную запись пользователя 3. Удаление учетной записи пользователя 4. Удаление всех учетных записей пользователей 5. Сортировка по возрастанию/А-Я 6. Сортировка по убыванию/Я-А 7. Поиск 0. Вернуться >>

Таблица 5 – Тестирование редактирования учетной записи пользователя

Исходные данные
1 1111 4 Почивалин Казаченко Максим Дмитриевич Сотрудник
Ожидаемый результат
ID#: 1 Фамилия: Казаченко Имя: Максим Отчество: Дмитриевич Отдел: Сотрудник ----------Меню редактирования учетных записей пользователей---------- 1. Фамилию Имя Отчество учетной записи пользователя 2. Отдел учетной записи пользователя 3. Пароль учетной записи пользователя 4. Все данные об учетной записи пользователя 0. Вернуться >>
Результат работы программы
ID#: 1 Фамилия: Казаченко Имя: Максим Отчество: Дмитриевич Отдел: Сотрудник ----------Меню редактирования учетных записей пользователей---------- 1. Фамилию Имя Отчество учетной записи пользователя 2. Отдел учетной записи пользователя 3. Пароль учетной записи пользователя 4. Все данные об учетной записи пользователя 0. Вернуться >>

Перед тем, как тестировать удаление учетной записи пользователя из списков, добавим пару учетных записей.

Таблица 6 – Тестирование удаления учетной записи пользователя

Исходные данные
1 да
Ожидаемый результат
ID#: 0 Фамилия: Бабанин Имя: Дмитрий Отчество: Сергеевич Отдел: Начальник ID#: 2 Фамилия: Почивалин Имя: Никита Отчество: Николаевич Отдел: Сотрудник ID#: 3 Фамилия: Лялихова Имя: Дарья Отчество: Сергеевна Отдел: Сотрудник ----------Меню управления учетными записями пользователя---------- 1. Добавить учетную запись пользователя 2. Редактировать учетную запись пользователя 3. Удаление учетной записи пользователя 4. Удаление всех учетных записей пользователей 5. Сортировка по возрастанию/А-Я 6. Сортировка по убыванию/Я-А 7. Поиск 0. Вернуться >>
Результат работы программы
ID#: 0 Фамилия: Бабанин Имя: Дмитрий Отчество: Сергеевич Отдел: Начальник ID#: 2 Фамилия: Почивалин Имя: Никита Отчество: Николаевич Отдел: Сотрудник ID#: 3 Фамилия: Лялихова Имя: Дарья

```
Отчество: Сергеевна
   Отдел: Сотрудник

----------Меню управления учетными записями пользователя----------
1.    Добавить учетную запись пользователя
2.    Редактировать учетную запись пользователя
3.    Удаление учетной записи пользователя
4.    Удаление всех учетных записей пользователей
5.    Сортировка по возрастанию/А-Я
6.    Сортировка по убыванию/Я-А
7.    Поиск
0.    Вернуться
>>
```

Таблица 7 – Тестирование удаления всех учетных записей пользователей

Исходные данные
да
Ожидаемый результат
ID#: 0 Фамилия: Бабанин Имя: Дмитрий Отчество: Сергеевич Отдел: Начальник ----------Меню управления учетными записями пользователя---------- 1. Добавить учетную запись пользователя 2. Редактировать учетную запись пользователя 3. Удаление учетной записи пользователя 4. Удаление всех учетных записей пользователей 5. Сортировка по возрастанию/А-Я 6. Сортировка по убыванию/Я-А 7. Поиск 0. Вернуться >>
Результат работы программы
ID#: 0 Фамилия: Бабанин Имя: Дмитрий Отчество: Сергеевич Отдел: Начальник ----------Меню управления учетными записями пользователя---------- 1. Добавить учетную запись пользователя 2. Редактировать учетную запись пользователя 3. Удаление учетной записи пользователя 4. Удаление всех учетных записей пользователей 5. Сортировка по возрастанию/А-Я 6. Сортировка по убыванию/Я-А 7. Поиск 0. Вернуться >>

Перед тем, как тестировать сортировку учетных записей пользователей, добавим пару учетных записей.

Таблица 8 – Тестирование сортировки по возростанию учетных записей пользователей

Исходные данные
2
Ожидаемый результат
ID#: 0 Фамилия: Бабанин Имя: Дмитрий Отчество: Сергеевич Отдел: Начальник ID#: 3 Фамилия: Королев Имя: Андрей Отчество: Викторович Отдел: Сотрудник ID#: 1 Фамилия: Почивалин Имя: Никита Отчество: Николаевич Отдел: Сотрудник ID#: 2 Фамилия: Черденко Имя: Василий Отчество: Алекстандрович Отдел: Сотрудник ----------Меню управления учетными записями пользователя---------- 1. Добавить учетную запись пользователя 2. Редактировать учетную запись пользователя 3. Удаление учетной записи пользователя 4. Удаление всех учетных записей пользователей 5. Сортировка по возрастанию/А-Я 6. Сортировка по убыванию/Я-А 7. Поиск 0. Вернуться >>
Результат работы программы
ID#: 0 Фамилия: Бабанин Имя: Дмитрий Отчество: Сергеевич Отдел: Начальник ID#: 3

```
 Фамилия: Королев
     Имя: Андрей
Отчество: Викторович
   Отдел: Сотрудник

     ID#: 1
 Фамилия: Почивалин
     Имя: Никита
Отчество: Николаевич
   Отдел: Сотрудник

     ID#: 2
 Фамилия: Черденко
     Имя: Василий
Отчество: Алекстандрович
   Отдел: Сотрудник

----------Меню управления учетными записями пользователя----------
1.    Добавить учетную запись пользователя
2.    Редактировать учетную запись пользователя
3.    Удаление учетной записи пользователя
4.    Удаление всех учетных записей пользователей
5.    Сортировка по возрастанию/А-Я
6.    Сортировка по убыванию/Я-А
7.    Поиск
0.    Вернуться
>>
```

Таблица 9 – Тестирование сортировки по убыванию учетных записей пользователей

Исходные данные
1
Ожидаемый результат
ID#: 3 Фамилия: Королев Имя: Андрей Отчество: Викторович Отдел: Сотрудник ID#: 2 Фамилия: Черденко Имя: Василий Отчество: Алекстандрович Отдел: Сотрудник ID#: 1 Фамилия: Почивалин Имя: Никита Отчество: Николаевич Отдел: Сотрудник ID#: 0 Фамилия: Бабанин

Имя: Дмитрий
Отчество: Сергеевич
Отдел: Начальник

----------Меню управления учетными записями пользователя----------
1. Добавить учетную запись пользователя
2. Редактировать учетную запись пользователя
3. Удаление учетной записи пользователя
4. Удаление всех учетных записей пользователей
5. Сортировка по возрастанию/А-Я
6. Сортировка по убыванию/Я-А
7. Поиск
0. Вернуться
>>

Результат работы программы

```
     ID#: 3
 Фамилия: Королев
     Имя: Андрей
Отчество: Викторович
   Отдел: Сотрудник

     ID#: 2
 Фамилия: Черденко
     Имя: Василий
Отчество: Алекстандрович
   Отдел: Сотрудник

     ID#: 1
 Фамилия: Почивалин
     Имя: Никита
Отчество: Николаевич
   Отдел: Сотрудник

     ID#: 0
 Фамилия: Бабанин
     Имя: Дмитрий
Отчество: Сергеевич
   Отдел: Начальник

----------Меню управления учетными записями пользователя----------
1.    Добавить учетную запись пользователя
2.    Редактировать учетную запись пользователя
3.    Удаление учетной записи пользователя
4.    Удаление всех учетных записей пользователей
5.    Сортировка по возрастанию/А-Я
6.    Сортировка по убыванию/Я-А
7.    Поиск
0.    Вернуться
>>
```

Таблица 10 – Тестирование поиска учетной записи пользователя

Исходные данные
П
Ожидаемый результат
ID#: 1 Фамилия: Почивалин Имя: Никита Отчество: Николаевич Отдел: Сотрудник
Результат работы программы
ID#: 1 Фамилия: Почивалин Имя: Никита Отчество: Николаевич Отдел: Сотрудник Для продолжения нажмите любую клавишу . . .

Таблица 11 – Тестирование добавления задания

Исходные данные
Построить график работы сотрудников 0 6 12 2017 До конца года
Ожидаемый результат
ID# задания: 0 ID# пользователя: 0 Дата выдачи задания: 4.12.2017 Дата планируемой сдачи: 6.12.2017 Дата сдачи задания: 0. 0. 0 Задание: Построить график работы сотрудников Примечание: До конца года ----------Меню управления заданийми---------- 1. Добавить задание 2. Редактировать задание 3. Удаление задания 4. Удаление всех заданий 5. Сортировка по возрастанию/А-Я 6. Сортировка по убыванию/Я-А 7. Поиск 8. Список просроченных заданий 9. Список заданий у заданного сотрудника 10. Список заданий, до истечения срока выполнения которых осталось X

дней 0. Вернуться >>
Результат работы программы
ID# задания: 0 ID# пользователя: 0 Дата выдачи задания: 4.12.2017 Дата планируемой сдачи: 6.12.2017 Дата сдачи задания: 0. 0. 0 Задание: Построить график работы сотрудников Примечание: До конца года ----------Меню управления заданиями---------- 1. Добавить задание 2. Редактировать задание 3. Удаление задания 4. Удаление всех заданий 5. Сортировка по возрастанию/А-Я 6. Сортировка по убыванию/Я-А 7. Поиск 8. Список просроченных заданий 9. Список заданий у заданного сотрудника 10. Список заданий, до истечения срока выполнения которых осталось X дней 0. Вернуться >> 1

Таблица 12 – Тестирование редактирования задания

Исходные данные
2 0 да 4
Ожидаемый результат
ID# задания: 0 ID# пользователя: 0 Дата выдачи задания: 4.12.2017 Дата планируемой сдачи: 6.12.2017 Дата сдачи задания: 4.12.2017 Задание: Построить график работы сотрудников Примечание: До конца года ----------Меню редактирования заданий---------- 1. Задание 2. Пренадлежность задания к пользователю 3. Планируемую дату сдачи 4. Дату сдачи задания 5. Примечание 6. Все данные о задании 0. Вернуться >>

Результат работы программы
ID# задания: 0 ID# пользователя: 0 Дата выдачи задания: 4.12.2017 Дата планируемой сдачи: 6.12.2017 Дата сдачи задания: 4.12.2017 Задание: Построить график работы сотрудников Примечание: До конца года ----------Меню редактирования заданий---------- 1. Задание 2. Пренадлежность задания к пользователю 3. Планируемую дату сдачи 4. Дату сдачи задания 5. Примечание 6. Все данные о задании 0. Вернуться >>

Тестирование удаления, сортировки и поиска задания пропускаем, так как они идентичны удалению, сортировке и поиску учетных записей пользователя.

Для тестирования вывода списка просроченных заданий, заданий конкретного пользователя и заданий, до истечения срока сдачи которых осталось 3 дня добавим еще пару заданий.

Таблица 13 – Тестирование списка просроченных заданий задания

Исходные данные
Ожидаемый результат
ID# задания: 0 ID# пользователя: 1 Дата выдачи задания: 10.11.2017 Дата планируемой сдачи: 11.11.2017 Дата сдачи задания: 0. 0. 0 Задание: Выгрузить весь груз в склад Примечание: Склад под номером 3 ID# задания: 2 ID# пользователя: 0 Дата выдачи задания: 11.11.2017 Дата планируемой сдачи: 12.11.2017 Дата сдачи задания: 0. 0. 0 Задание: Наказать Примечание: Сотрудника под №1

ID# задания: 3
ID# пользователя: 3
Дата выдачи задания: 26.11.2017
Дата планируемой сдачи: 27.11.2017
Дата сдачи задания: 0. 0.   0
Задание: Убить Била
Примечание: Секретно

ID# задания: 4
ID# пользователя: 2
Дата выдачи задания: 26.11.2017
Дата планируемой сдачи: 30.11.2017
Дата сдачи задания: 0. 0.   0
Задание: Зашифровать коды запуска ядерных ракет
Примечание: Секретно

Результат работы программы

```
           ID# задания: 0
      ID# пользователя: 1
   Дата выдачи задания: 10.11.2017
Дата планируемой сдачи: 11.11.2017
    Дата сдачи задания:  0. 0.   0
               Задание: Выгрузить весь груз в склад
            Примечание: Склад под номером 3

           ID# задания: 2
      ID# пользователя: 0
   Дата выдачи задания: 11.11.2017
Дата планируемой сдачи: 12.11.2017
    Дата сдачи задания:  0. 0.   0
               Задание: Наказать
            Примечание: Сотрудника под №1

           ID# задания: 3
      ID# пользователя: 3
   Дата выдачи задания: 26.11.2017
Дата планируемой сдачи: 27.11.2017
    Дата сдачи задания:  0. 0.   0
               Задание: Убить Била
            Примечание: Секретно

           ID# задания: 4
      ID# пользователя: 2
   Дата выдачи задания: 26.11.2017
Дата планируемой сдачи: 30.11.2017
    Дата сдачи задания:  0. 0.   0
               Задание: Зашифровать коды запуска ядерных ракет
            Примечание: Секретно

Для продолжения нажмите любую клавишу . . .
```

Таблица 14 – Тестирование списка просроченных заданий задания

Исходные данные
3
Ожидаемый результат
ID# задания: 3 ID# пользователя: 3 Дата выдачи задания: 26.11.2017 Дата планируемой сдачи: 27.11.2017 Дата сдачи задания: 0. 0.   0 Задание: Убить Била Примечание: Секретно ID# задания: 6 ID# пользователя: 3 Дата выдачи задания: 26.11.2017 Дата планируемой сдачи: 31.12.2017 Дата сдачи задания: 0. 0.   0 Задание: Выполнить годичный бухгалтерский учет компании Примечание: Учет разделить на отделы
Результат работы программы
ID# задания: 3 ID# пользователя: 3 Дата выдачи задания: 26.11.2017 Дата планируемой сдачи: 27.11.2017 Дата сдачи задания: 0. 0.   0 Задание: Убить Била Примечание: Секретно ID# задания: 6 ID# пользователя: 3 Дата выдачи задания: 26.11.2017 Дата планируемой сдачи: 31.12.2017 Дата сдачи задания: 0. 0.   0 Задание: Выполнить годичный бухгалтерский учет компании Примечание: Учет разделить на отделы Для продолжения нажмите любую клавишу . . .

Таблица 15 – Тестирование списка просроченных заданий задания

Исходные данные
Ожидаемый результат
ПО ВАШЕМУ ЗАПРОСУ ДАННЫХ НЕ СУЩЕСТВУЕТ! ПОПРОБУЙТЕ ЕЩЕ РАЗ!
Результат работы программы
ПО ВАШЕМУ ЗАПРОСУ ДАННЫХ НЕ СУЩЕСТВУЕТ! ПОПРОБУЙТЕ ЕЩЕ РАЗ! Для продолжения нажмите любую клавишу . . .

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Программа представляет собой консольное приложение. Ввод данных осуществляется с клавиатуры. Подтверждение ввода клавишей «Enter». Символьная строка воспринимается как «0», если поле воспринимает числа.

Работа программы начинается с вывода в консоль информации о программе, учреждении, которая его спроектировала, и об участниках проекта (рисунок 5.1).

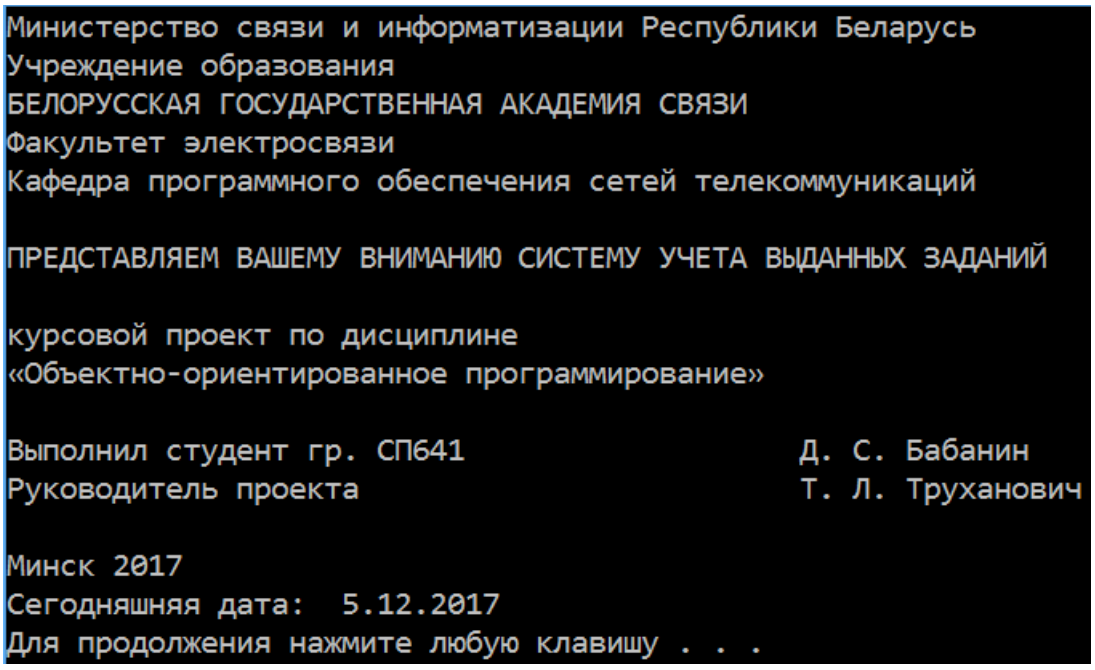

Рисунок 5.1 – Информация о программе, учреждении, которая его спроектировала, и об участниках проекта.

После отображается форма авторизации системы, которая требует ввода логина и пароля (рисунок 5.2). Логин представляет собой идентификатор пользователя, у администратора заранее зарезервирован 0, с паролем 0000. Обычный пользователя должен быть изначально зарегистрирован администратором. Администратор должен выдан логин и пароль от личной учетной записи (значение пароля устанавливается по запросу сотрудника, которому принадлежит учетная запись).

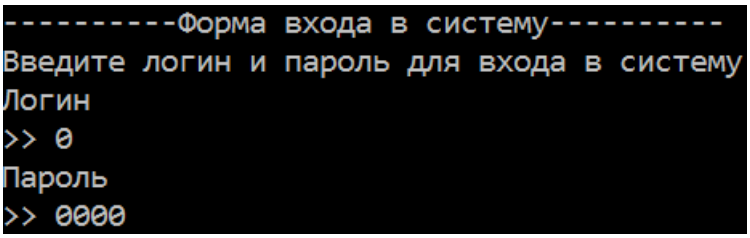

Рисунок 5.2 – Пример авторизации в систему

После авторизации в системе, отображается информация об учетной записи пользователя (рисунок 5.3).

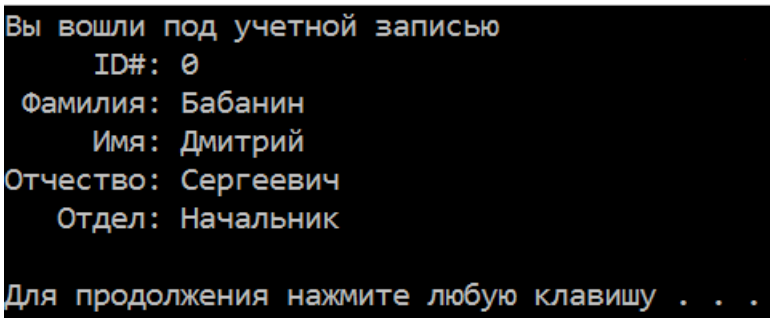

Рисунок 5.3 – Информация об учетной записи пользователя

Если же при вводе логина или пароля возникла ошибка, то программа выводит соответствующее сообщение (рисунки 5.4, 5.5).

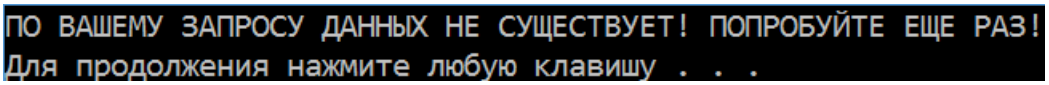

Рисунок 5.4 – Сообщение, которое выводиться при вводе логина, которого не существует в списках учетных записей пользователей.

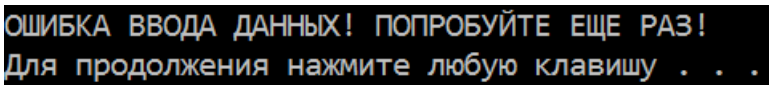

Рисунок 5.5 – Сообщение, которое выводиться при вводе не верного пароля.

После ввода логина и пароля учетной записи администратора, выводиться «Главное меню администратора». Для перехода в выбранный пункт меню необходимо ввести соответствующее число в поле ввода (рисунок 5.6).

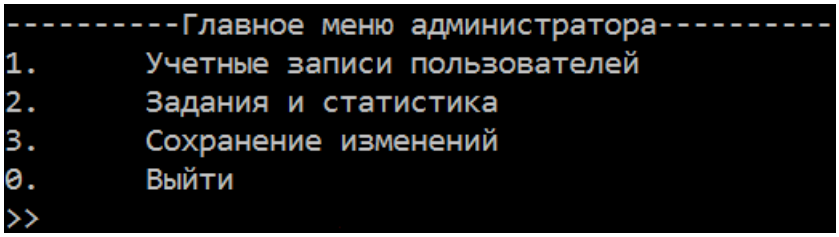

Рисунок 5.6 – Главное меню администратора

При вводе числа, который не соответствует пунктам меню, выводится соответствующее сообщение (рисунок 5.7).

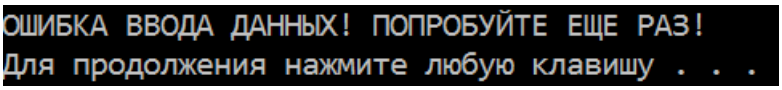

Рисунок 5.7 – Ошибка при выборе пункта меню

При вводе символьной строки происходит выход из программа, так как она воспринимается как 0. *В дальнейшем все меню имеют такие свойства.*

Выберем пункт меню «Учетные записи пользователей», введя «1» (рисунок 5.8).

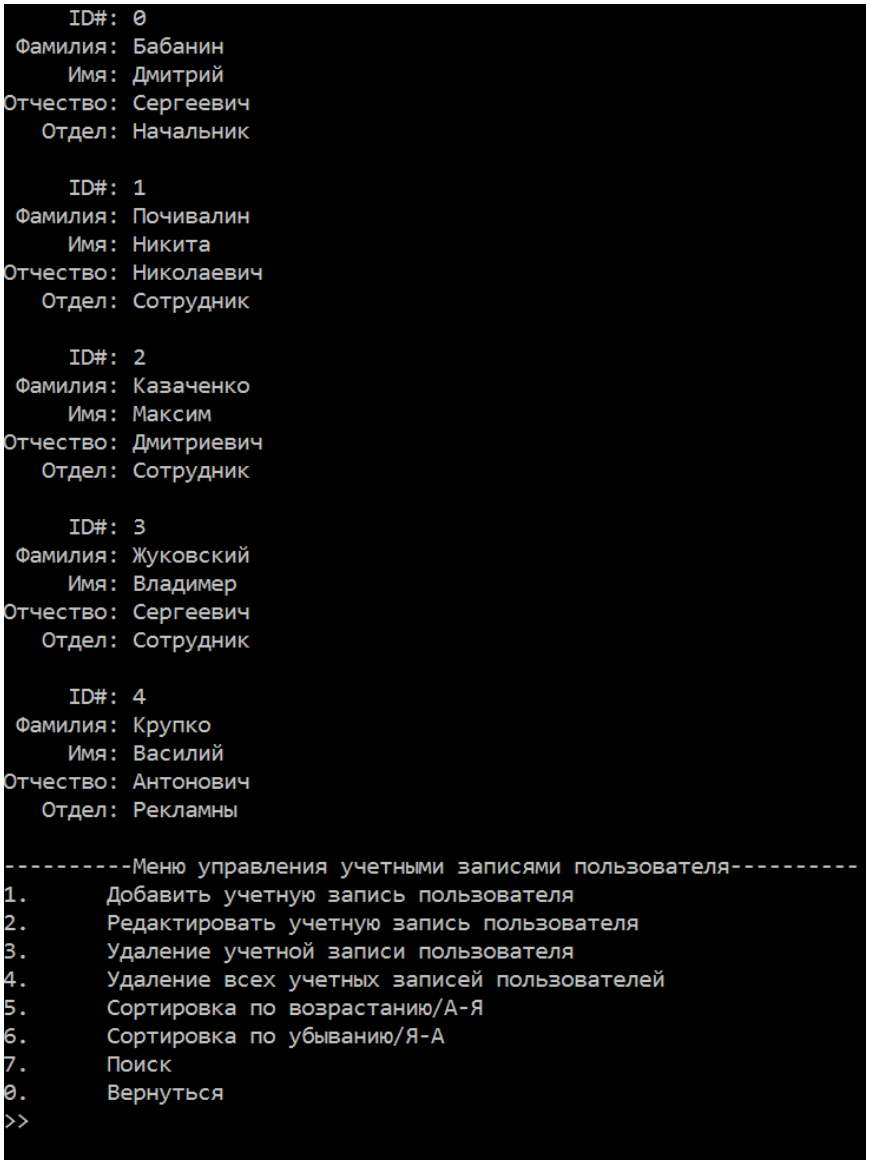

Рисунок 5.8 – Список учетных записей пользователей и «Меню управления учетными записями пользователя»

Выводиться список учетных записей, отсортированный по возрастанию идентификатора учетной записи, и «Меню управления учетными записями пользователя». Учетная запись отображается в виде анкеты содержащая в себе следующую информацию: идентификатор пользователя, ФИО и отдел, где работает, или должность, которую занимает на предприятии. Каждая учетная запись разделена пустой строкой. Для удобства добавлено еще несколько учетных записей.

Выберем пункт «Добавить учетную запись пользователя», введя «1» (рисунок 5.9).

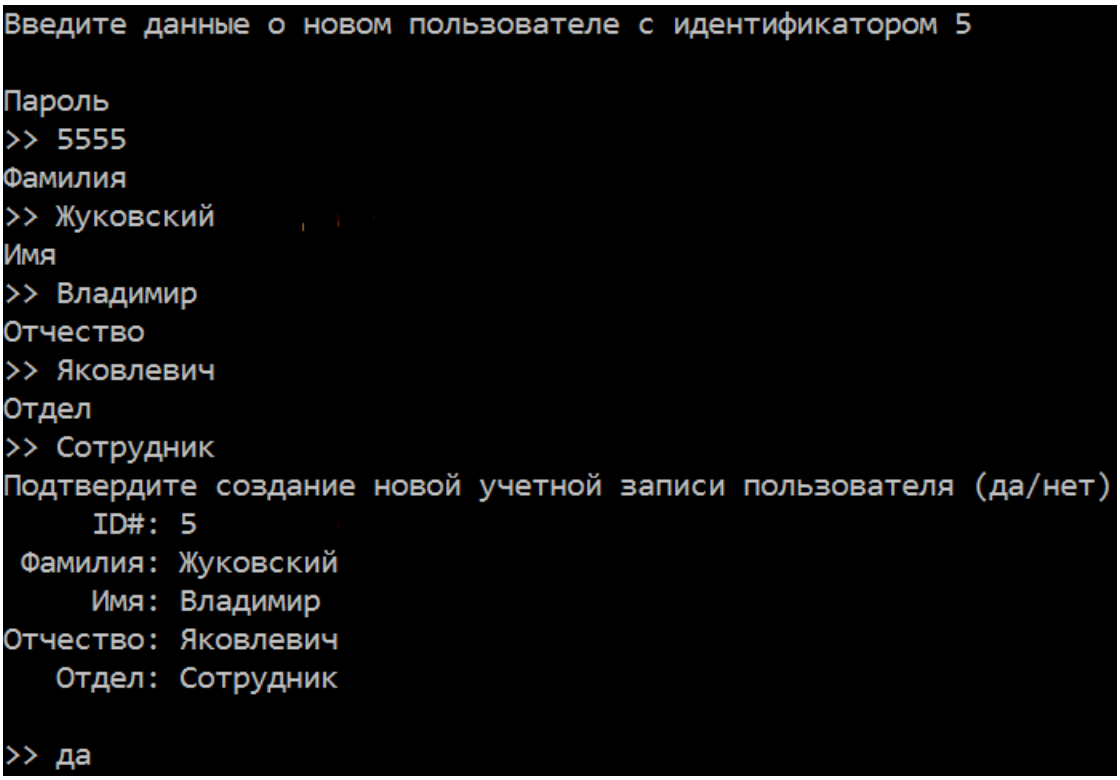

Рисунок 5.9 – Форма добавления новой учетной записи пользователя

Выводиться форма добавления новой учетной записи пользователя. Идентификатор пользователя генерируется программой, так что администратору не нужно тратить время на придумывание уникальной последовательности чисел для учетной записи пользователя. Пароль, ФИО, Отдел заполняется администратором. Форма ввода пароля позволяет вводить символьные строки любой сложности. Формы ввода ФИО и отдела не позволяют вводить символьных строк, содержащие пробелы, формой воспринимается только последовательность символов перед первым пробелом. Сразу же отображается форма подтверждения создания новой учетной записи пользователя с введенными данными для проверки

правильности введенной информации. Если в форму ввести «нет», то программа проигнорирует введенные данные и перейдет обратно в «Меню управления учетными записями пользователя». Если «да», то программа перейдет обратно в «Меню управления учетными записями пользователя» с обновленным списком аккаунтов (рисунок 5.10).

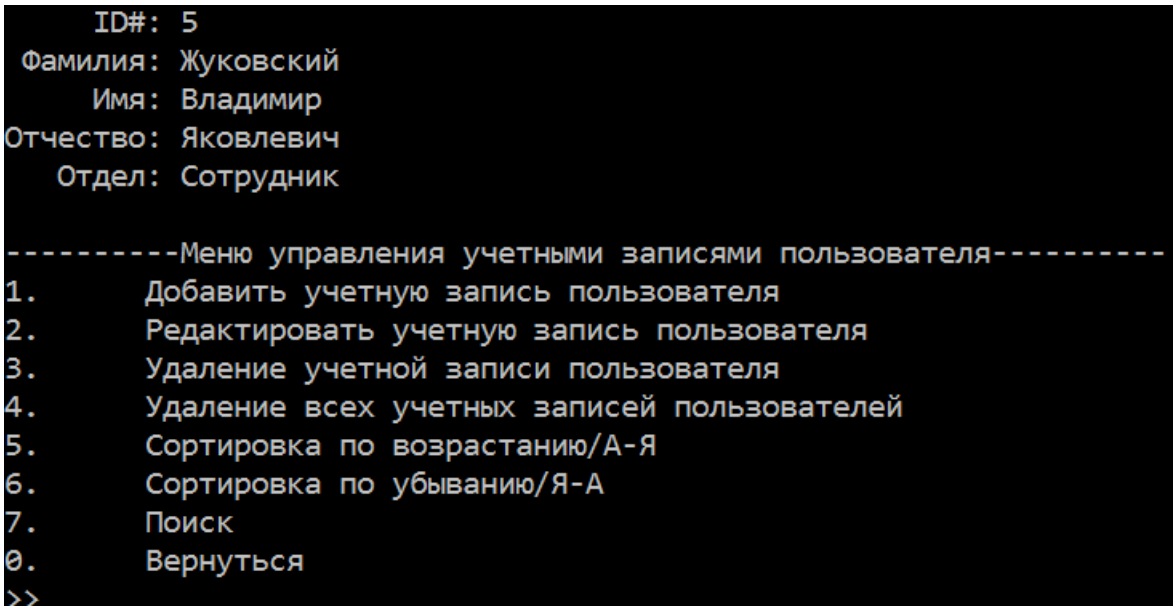

Рисунок 5.10 – Меню управления учетными записями пользователя с обновленным списком аккаунтов (без остальных учетных записей для экономии бумаги)

Если ввести любую символьную строку не эквивалентную словам «да» или «нет», то программа отобразит соответствующее сообщение (рисунок 5.11). *В дальнейшем все формы подтверждения работают идентично, кроме подтверждения о редактировании учетной записи пользователя.*

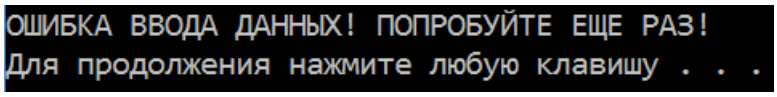

Рисунок 5.11 – Сообщение при некорректном вводе подтверждения добавления новой учетной записи пользователя

В «Меню управления учетными записями пользователя» выберем пункт «Редактировать учетную запись пользователя», введя «2» (рисунок 5.12).

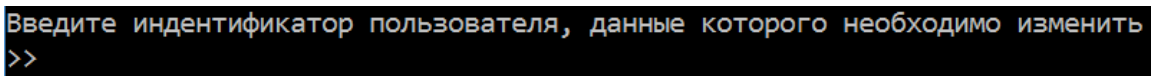

Рисунок 5.12 – Форма поиска учетной записи пользователя

Выводиться форма поиска учетной записи пользователя, которая просит идентификатор учетной записи пользователя, которую необходимо отредактировать. Введем идентификатор пользователя, которого мы добавили «5» (рисунок 5.13).

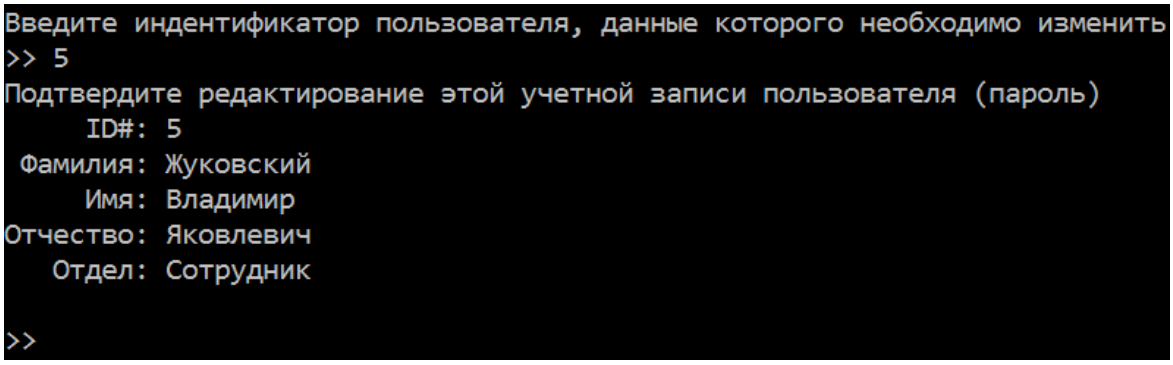

Рисунок 5.13 – Форма подтверждения редактирования учетной записи пользователя.

Выводиться форма подтверждения редактирования учетной записи пользователя, которая просит пароль для редактирования.

При вводе идентификатора, учетной записи которого не существует, выводиться соответствующее сообщение (рисунок 5.14).

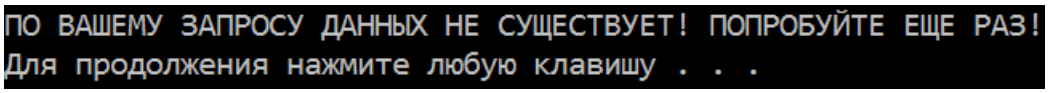

Рисунок 5.14 – Сообщение при не корректном вводе идентификатора учетной записи пользователя.

При вводе не правильного пароля, программа выводит соответствующее сообщение (рисунок 5.15).

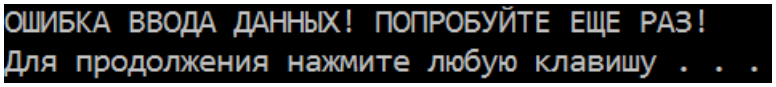

Рисунок 5.15 – Сообщение при не корректном вводе пароля

Вводим в форму подтверждения редактирования учетной записи пользователя «5555» (рисунок 5.16).

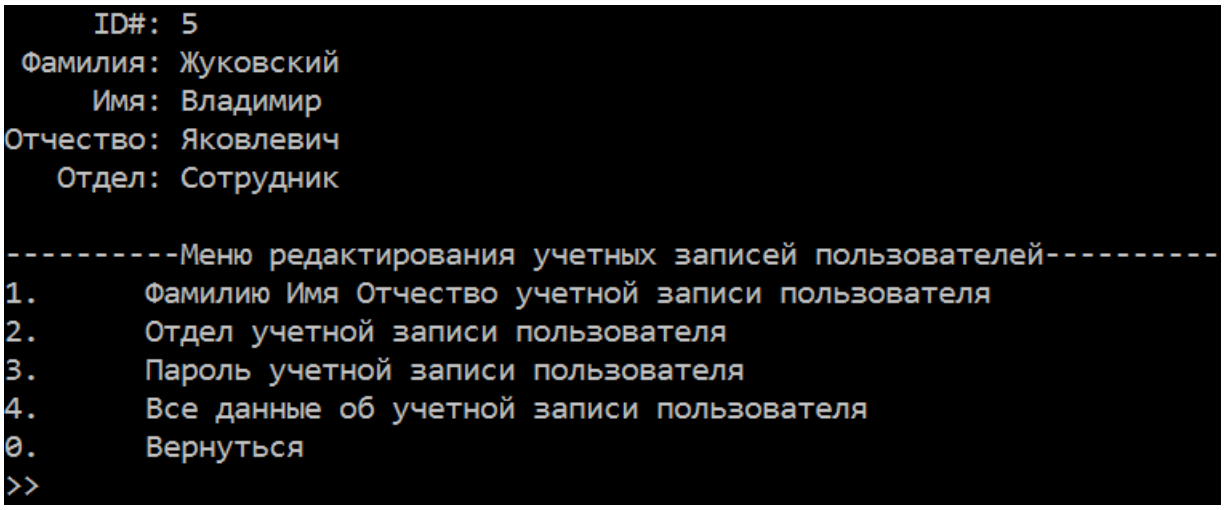

Рисунок 5.16 – Меню редактирования учетной записи пользователя

Выводиться меню редактирования учетной записи пользователя. Каждый пункт соответствует определенному виду редактирования, после вывода форма требует новые данные для определенного поля учетной записи (рисунки 5.17, 5.18, 5.19, 5.20).

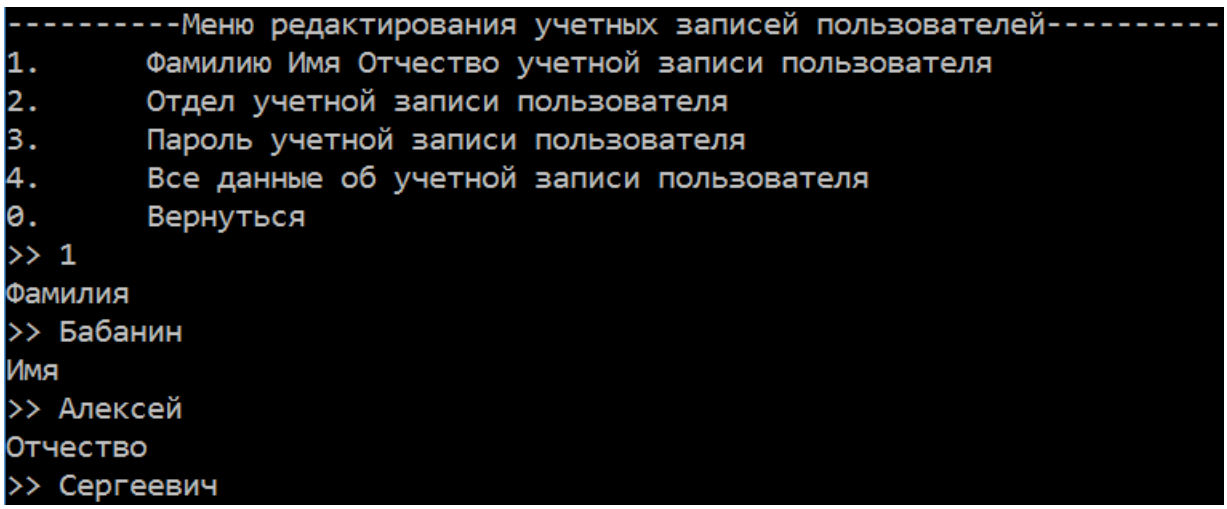

Рисунок 5.17 – Форма редактирования ФИО

```
----------Меню редактирования учетных записей пользователей----------
1.      Фамилию Имя Отчество учетной записи пользователя
2.      Отдел учетной записи пользователя
3.      Пароль учетной записи пользователя
4.      Все данные об учетной записи пользователя
0.      Вернуться
>> 2
Отдел
>> Рекламный
```

Рисунок 5.18 – Форма редактирования отдела

```
----------Меню редактирования учетных записей пользователей----------
1.      Фамилию Имя Отчество учетной записи пользователя
2.      Отдел учетной записи пользователя
3.      Пароль учетной записи пользователя
4.      Все данные об учетной записи пользователя
0.      Вернуться
>> 3
Новый пароль
>> Бабанин
```

Рисунок 5.19 – Форма редактирования пароля

```
----------Меню редактирования учетных записей пользователей----------
1.      Фамилию Имя Отчество учетной записи пользователя
2.      Отдел учетной записи пользователя
3.      Пароль учетной записи пользователя
4.      Все данные об учетной записи пользователя
0.      Вернуться
>> 4
Пароль
>> 5555
Фамилия
>> Бабанин
Имя
>> Алексей
Отчество
>> Сергеевич
Отдел
>> Сотрудник
```

Рисунок 5.20 – Форма редактирования всех данных

В «Меню управления учетными записями пользователя» выберем пункт «Удаления учетной записи пользователя», введя «3» (рисунок 5.21, 5,22).

```
ВВедите индентификатор пользователя, данные которого необходимо удалить
>> 5
Подтвердите удаление учетной записи пользователя и всех его заданий (да/нет)
     ID#: 5
 Фамилия: Бабанин
     Имя: Алексей
Отчество: Сергеевич
   Отдел: Сотрудник

>> да
```

Рисунок 5.22 – Форма поиска учетной записи для удаления, форма подтверждения для удаления

После ввода идентификатора и подтверждения программа возвращается в «Меню управления учетными записями пользователя».

В главном меню администратора выберем пункт «Задания и статистика», введя «2» (рисунок 5.23).

```
----------Меню управления заданийми----------
1.      Добавить задание
2.      Редактировать задание
3.      Удаление задания
4.      Удаление всех заданий
5.      Сортировка по возрастанию/А-Я
6.      Сортировка по убыванию/Я-А
7.      Поиск
8.      Список просроченных заданий
9.      Список заданий у заданного сотрудника
10.     Список заданий, до истечения срока выполнения которых осталось X дней
0.      Вернуться
>>
```

Рисунок 2.23 – Меню управления заданиями (администратора)

Выводиться список заданий и «Меню управления заданиями», на рисунке 2.23 список заданий отсутствует, так как список пуст.

Выберем пункт «Добавить задание», введя «1» (рисунок 2.24).

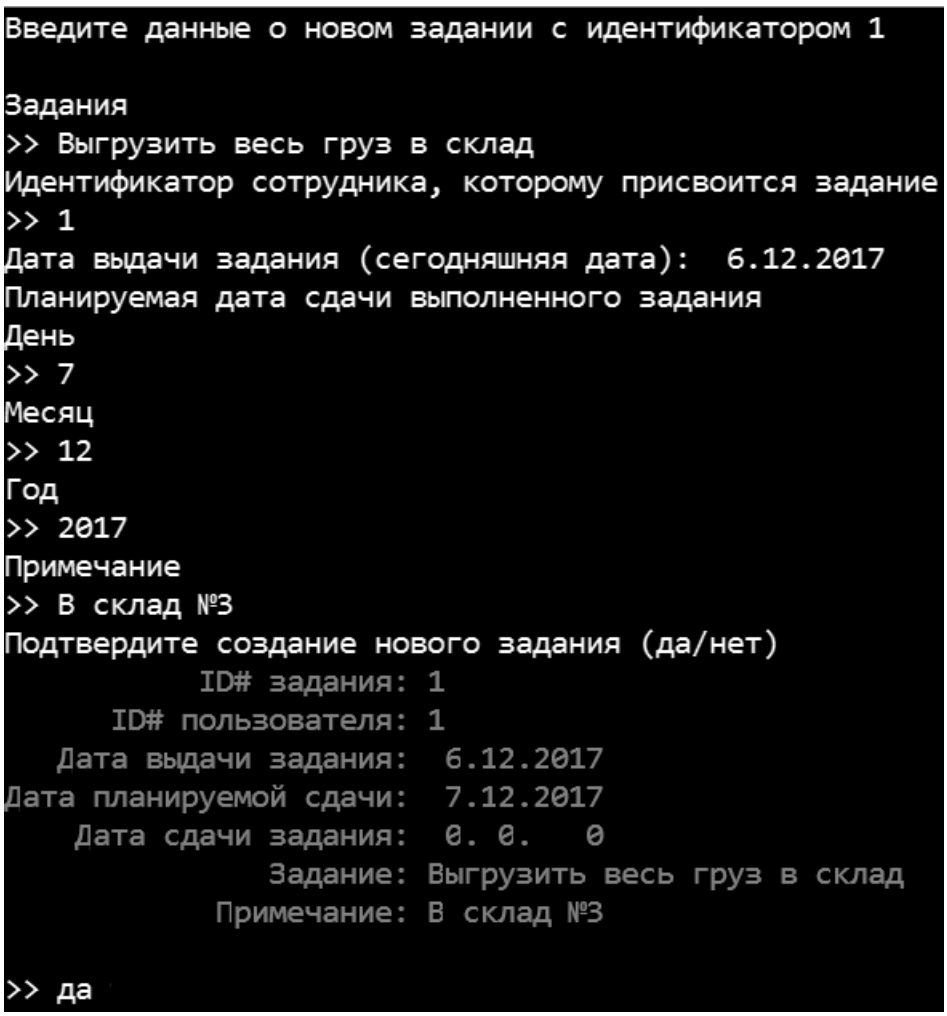

Рисунок 2.24 – Форма добавления задания

Выводиться форма добавления нового задания. Идентификатор задания генерируется программой, так что администратору не нужно тратить время на придумывание уникальной последовательности чисел для задания. Задание, идентификатор, которому будет принадлежать задание, планируемую дату сдачи, примечание заполняется администратором. Дата выдачи задания (дата создания нового задания) генерируется программой для того, чтобы уберечь базу данных от ложных данных. Форма ввода задания и примечания позволяет вводить символьные строки любой сложности. Формы ввода идентификатора, которому будет принадлежать задание, и планируемая дата сдачи позволяет вводить только цифры. Сразу же отображается форма подтверждения создания нового задания с введенными данными для проверки правильности введенной информации. Если в форму ввести «нет», то программа проигнорирует введенные данные и перейдет обратно в «Меню управления заданиями». Если «да», то программа перейдет

обратно в «Меню управления заданиями» с обновленным списком аккаунтов (рисунок 5.25).

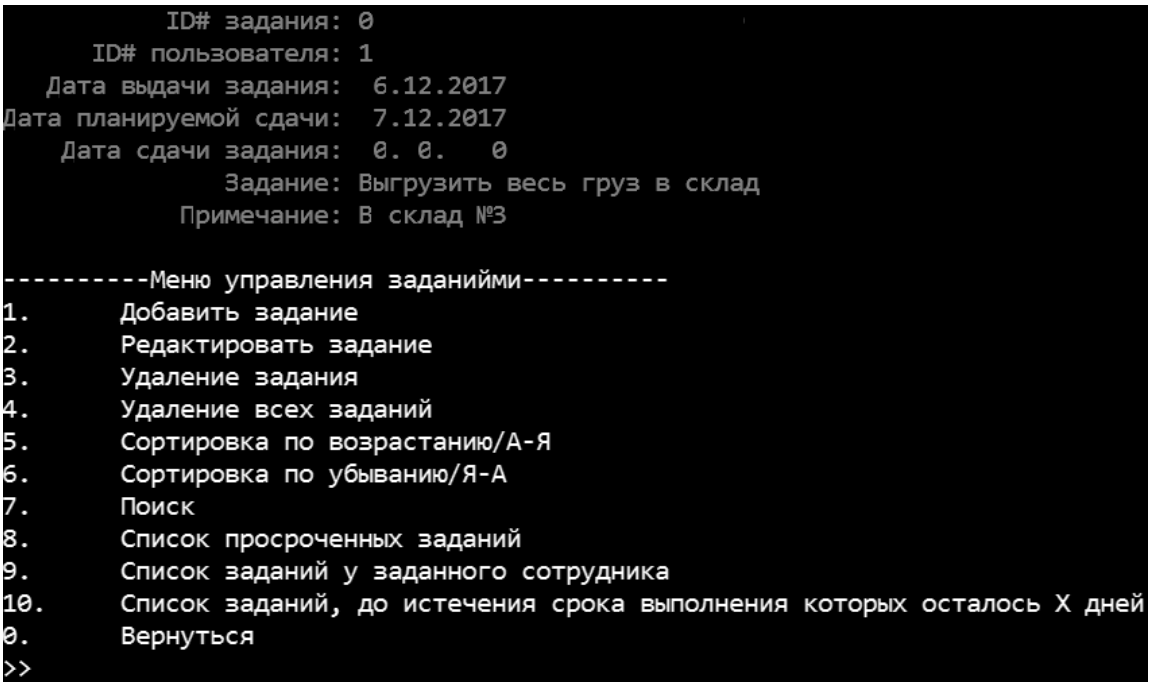

Рисунок 5.25 – Меню управления заданиями с обновленным списком аккаунтов.

Если дата планируемой сдачи задания будет меньше чем дата выдачи задания, то отобразиться соответствующее сообщение (рисунок 5.26).

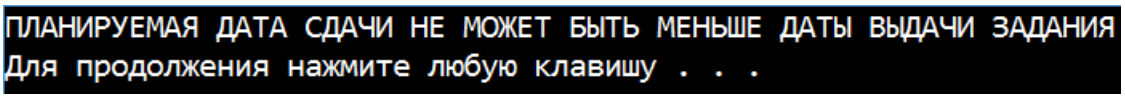

Рисунок 5.26 – Сообщение при некорректном вводе планируемой даты сдачи

Пункты «Редактировать задание», «Удаление задания», «Удаление всех заданий», «Сортировка по возростанию/А-Я», «Сортировка по убыванию/Я-А», «Поиск» реализованы также, как и функционал «Меню управления учётными записями пользователей».

Перед тем, как воспользоваться остальными функциями «Меню управления заданиями» добавим еще несколько заданий.

Выберем пункт меню «Список просроченных заданий», введя «8» (рисунок 5.27).

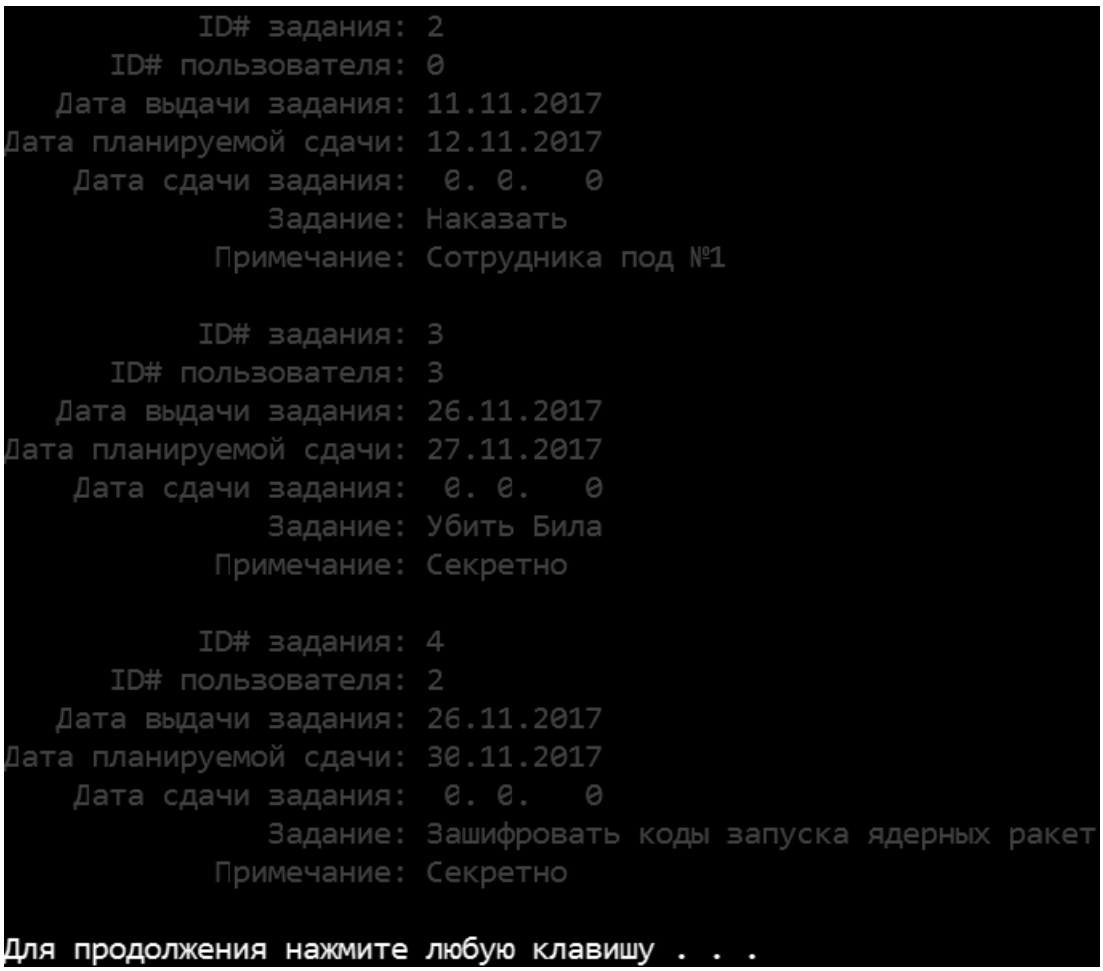

Рисунок 5.27 – Список просроченных заданий

Отображаются все задания, срок выполнения которых меньше сегодняшней даты (выделены красным). Задания отображаются красным цветом в таком случае, их можно увидеть и в общем списке заданий.

Выберем пункт «Список заданий у заданного пользователя», введя «9» (рисунок 5.28).

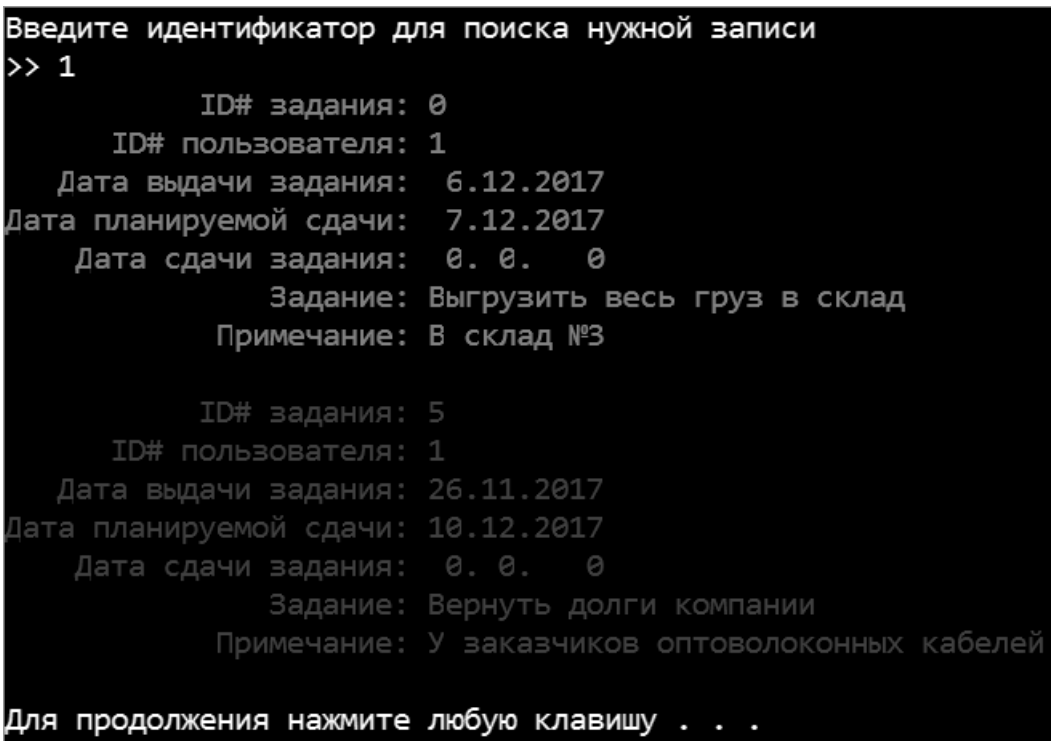

Рисунок 5.28 – Список заданий у заданного пользователя

Отображается форма ввода поиска заданий по введенному идентификатору учетной записи пользователя. После ввода идентификатора выводиться список заданий введенного идентификатора пользователя. Задание под идентификатором 0 отображается желтым цветом, что обозначает, что задание должно быть сдано в течении 3 дней или меньше. А задание под идентификатором 5 отображается синим цветом, что обозначает, что задание должно быть сдано в течении 6 дней или меньше. Здания отображающиеся белым цветом должны быть сданы в течении времени больше чем 6 дней или уже выполненые.

Выберем пункт «Список заданий, до истечения срока выполнения которых осталось 3 дня», введя «10» (рисунок 5.29).

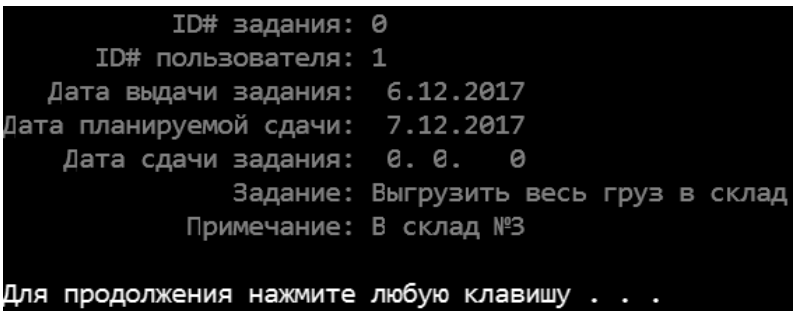

Рисунок 5.29 – Список заданий, до истечения срока выполнения которых осталось 3 дня

Отображается список заданий, до истечения срока выполнения которых осталось 3 дня (выделены желтым).

Войдем в систему под любой другой учетной записью пользователя (рисунок 5.30).

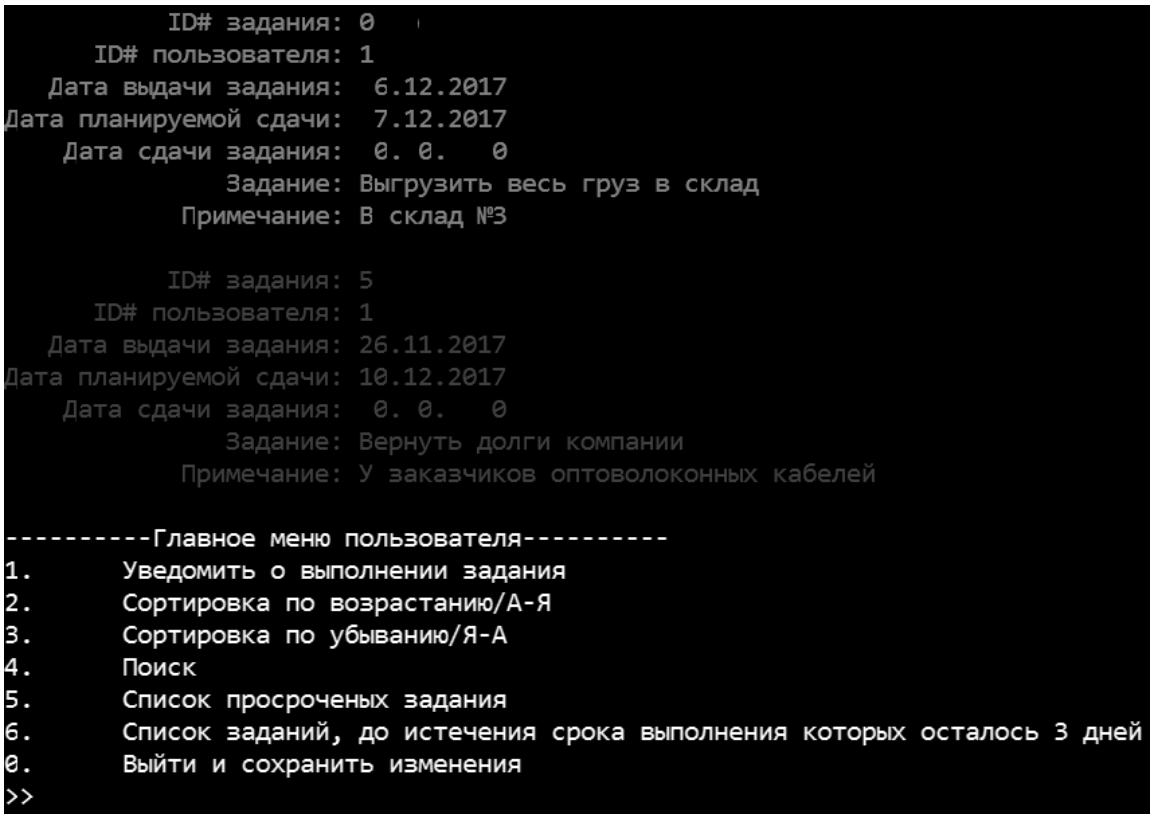

Рисунок 5.30 – Главное меню пользователя

Отображается список заданий пользователя и «Главное меню пользователя». В списке собраны только те задания, которые принадлежат пользователя, под учетной записью которого он вошел в систему и отображаются тоже красными, желтыми, синими и белыми цветами.

Выберем пункт «Уведомить о выполнении задания», введя «1» (рисунок 5.31).

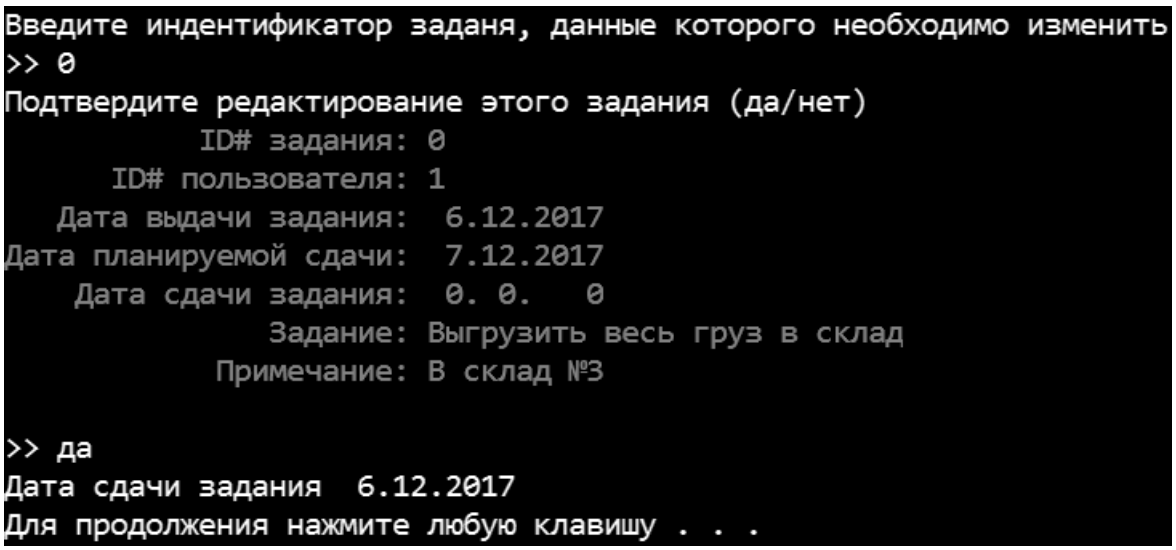

Рисунок 5.31 – Форма для уведомления о выполнении задания

Отображается форма для уведомления о выполнении задания, которая состоит из 2 форма: форма поиска задания, форма подтверждения изменения данных о задании. После подтверждения дата сдачи задания меняется на сегодняшнюю дату. После возвращаемся в «Главное меню пользователя» с обновленным списком заданий.

Пункты «Сортировка по возрастанию/А-Я», «Сортировка по убыванию/Я-А», «Поиск», «Список просроченных заданий», «Список заданий, до истечения срока выполнения которых осталось 3 дня» реализованы также, как и функционал «Меню управления заданиями» у администратора, отличие лишь в том, что программа работает только с заданиями, которые принадлежат пользователя, под учетной записью которого он вошел в систему.

При выборе пункта «Выход и сохранение изменений» программа записывает изменения в специальный файл и завершает работу программы.

ЗАКЛЮЧЕНЕ

Назначение программы заключается в том, чтобы упростить координацию работу предприятия, ускорить проверку выполнения требований к сотрудникам предприятия, улучает оперативность работы предприятия, уменьшает количество ошибок при работе над проектами в предприятии, упрощение распределения нагрузки на сотрудников, помогает узнать, над чем работал тот или иной сотрудник на протяжении всего времени работы на предприятии.

Программа может эксплуатироваться в любой компании и даже для обычного человека, который хочет отрегулировать свои действия на протяжении большого количество времени.

Важно особенностью использования программы на предприятии это экономия времени.

Из минусов данной программы можно выделить пару пунктов. Например, работа в данной программе приходится производить только на персональных компьютерах (ПК), вследствие чего удаленно ничего нельзя будет сделать с файлами данных сотрудников. Так же данные можно повредить, если обращаться с программой очень небрежно. Кроме того, стоит учитывать возможность взлома или повреждения программы, если ваш компьютер подхватил вирус.

Программу можно развить следующим образом: для надежного хранения данных для программы лучше использовать существующие СУБД, для много поточного доступа разработать программу на личном сервере предприятия при этом увеличивается защита от нежелательного доступа к данным. Можно дополнительно разработать софт для защиты данных от пользователей не имеющее отношение к предприятию. Такая программа уже будет web-приложением, так как к нему будут иметь доступ сотрудники предприятия которые имеют свои аккаутны для доступа к данным.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Павловская Т.А. С/С++. Программирование на языке высокого уровня – СПб.: Питер, 2006. – 461 с.: ил.
2. Павловская Т.А.,Щупак Ю.А. С/С++. Структурное программирование: Практикум – СПб.: Питер, 2002. – 240 с.: ил.
3. Введение в язык С++, Бьярн Страуструп, 1995 г., электронный учебник.
4. Подбельский В.В. Язык Си++: Учеб. Пособие.-5-е изд.-М.:Финансы и статистика, 2004. – 56-с.
5. Лафоре Р. Объектно-ориентированнное программирование в С++; Пер с англ. – Спб.:Питер, 2007. – 928 с.
6. Stack Overflow на русском [Электронный ресурс]. – Режим доступа: http://ru.stackoverflow.com/
7. Stack Overflow [Электронный ресурс]. – Режим доступа: http://stackoverflow.com/
8. CyberForum [Электронный ресурс]. – Режим доступа: http://cyberforum.ru/
9. Computer Science Center – [Электронный ресурс]. – Режим доступа: https://www.youtube.com/channel/UC0YHNueF-3Nh3uQT0P4YQZw

Приложение А

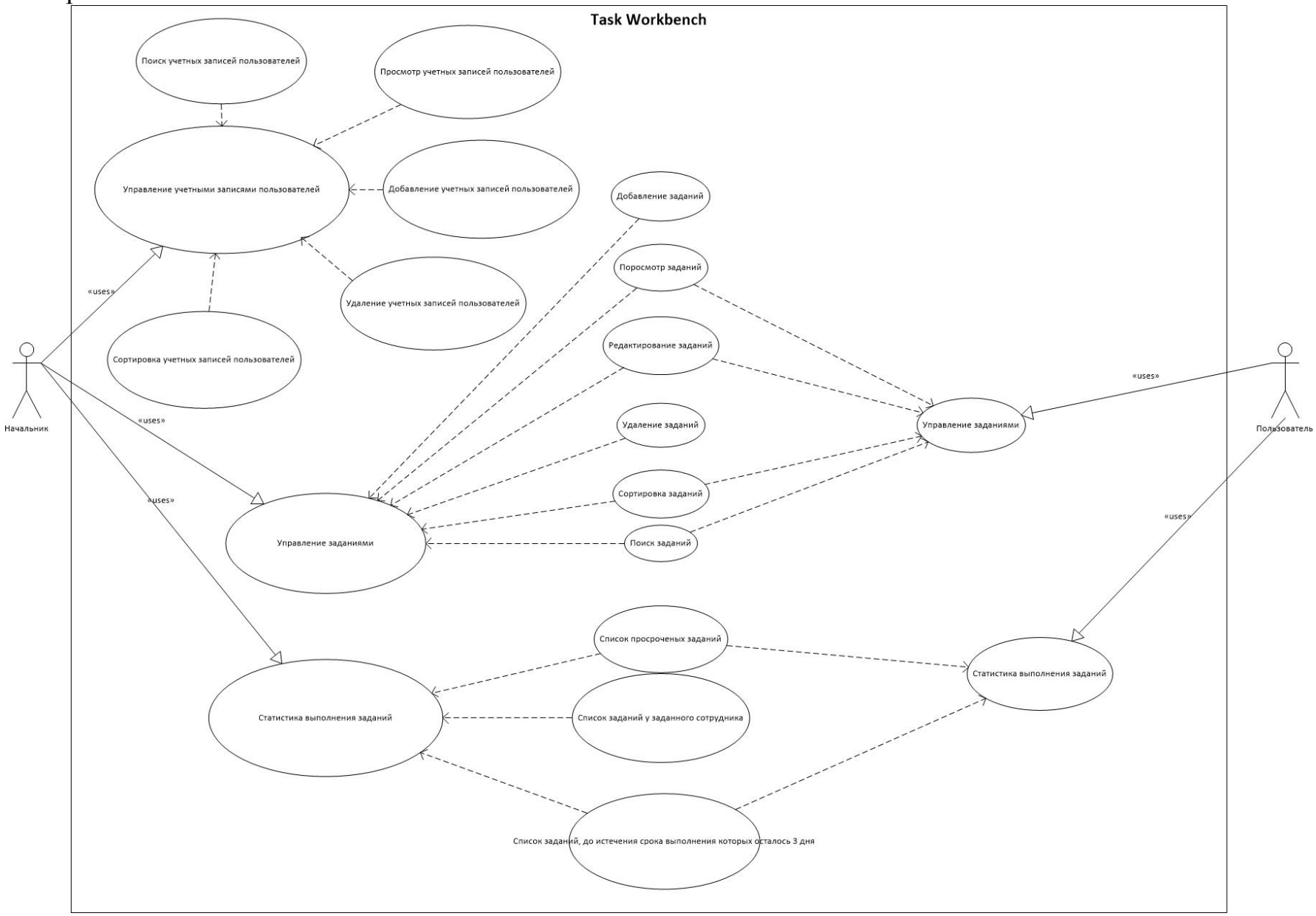

Рисунок А.1 – Диаграмма вариантов использования

Приложение Б

Рисунок Б.1 – Диаграмма классов

ПРИЛОЖЕНИЕ В

Табица 16 – Исходный код в файле main.cpp

Main.cpp

```
#include <iostream>
#include <list>
#include "./Screens/LoginAndPasswordScreen.h"
#include "./Screens/AdministratorScreen.h"
#include "./Screens/UserScreen.h"
#include "./Controllers/ControllerIOFStream.h"
#include "./Data_Classes/UserAccount.h"
#include "./Data_Classes/Task.h"
using namespace std;

int main() {
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	Date todayDate(Date::getTodayDate());
	cout << "Министерство связи и информатизации Республики Беларусь\n"
		<< "Учреждение образования\n"
		<< "БЕЛОРУССКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ СВЯЗИ\n"
		<< "Факультет электросвязи\n"
		<< "Кафедра программного обеспечения сетей
телекоммуникаций\n\n"
		<< "ПРЕДСТАВЛЯЕМ ВАШЕМУ ВНИМАНИЮ СИСТЕМУ УЧЕТА ВЫДАННЫХ
ЗАДАНИЙ\n\n"
		<< "курсовой проект по дисциплине\n"
		<< "«Объектно-ориентированное программирование»\n\n" <<
setw(45) << left
		<< "Выполнил студент гр. СП641" << right << "Д. С.
Бабанин\n"
		<< setw(45) << left << "Руководитель проекта" << right <<
"Т. Л. Труханович\n\n"
		<< left << "Минск 2017\n"
		<< "Сегодняшняя дата: " << todayDate << endl;
	system("pause");

	list<UserAccount> ListUserAccounts;
	ListUserAccounts = readFile(UserAccount::PATH_USER_ACCOUNTS,
		ListUserAccounts);
	list<Task> ListTasks;
	ListTasks = readFile(Task::PATH_TASKS, ListTasks);
	list<UserAccount>::iterator activeAccount =

	LoginAndPasswordScreen::instance(ListUserAccounts).showLoginAndPasswordS
creen();
	if ((*activeAccount).getUserId() == 0) {
		AdministratorScreen::instance(ListUserAccounts,
ListTasks).showAdministratorScreen();
		return 0;
	}
	UserScreen::instance(ListTasks, activeAccount).showUserScreen();
	return 0;
}
```

Табица 17 – Исходный код в файле Date.h

Date.h

```
#ifndef DATA_CLASSES_DATE_H_
#define DATA_CLASSES_DATE_H_
#include <string>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <ctime>
#include  "../Controllers/ControllerIStream.h"
#include "../Screens/ExceptionScreen.h"
using namespace std;

class Date {
private:
	short int day;
	short int month;
	short int year;
public:
	Date();
	Date(short int, short int, short int);
	Date(const Date&);
	~Date();
	void setDay(short int);
	void setMonth(short int);
	void setYear(short int);
	short int getDay();
	short int getMonth();
	short int getYear();
	string toString();
	Date& operator=(const Date&);
	int operator-(const Date&);
	int comparisonDate(const Date&);
	friend bool operator<(const Date&, const Date&);
	friend bool operator==(const Date&, const Date&);
	friend istream &operator>>(istream&, Date&);
	friend ostream &operator<<(ostream&, Date&);
	friend ifstream &operator>>(ifstream&, Date&);
	friend ofstream &operator<<(ofstream&, Date&);
	static Date getTodayDate();
};

#endif /* DATA_CLASSES_DATE_H_ */
```

Табица 18 – Исходный код в файле Date.cpp

Date.cpp

```
#include "Date.h"

Date::Date() :
		day(0), month(0), year(0) {
}

Date::Date(short int day, short int month, short int year) :
		day(day), month(month), year(year) {
}

Date::Date(const Date& obj) :
		day(obj.day), month(obj.month), year(obj.year) {
}

Date::~Date() {
```

```
}

void Date::setDay(short int day) {
	this->day = day;
}

void Date::setMonth(short int month) {
	this->month = month;
}

void Date::setYear(short int year) {
	this->year = year;
}
short int Date::getDay() {
	return this->day;
}

short int Date::getMonth() {
	return this->month;
}

short int Date::getYear() {
	return this->year;
}

string Date::toString() {
	string str;
	str.append("[Date@");
	str.append("day=");
	str.append(to_string(this->day));
	str.append(",month=");
	str.append(to_string(this->month));
	str.append(",year=");
	str.append(to_string(this->year));
	str.append("]");
	return str;
}

Date& Date::operator=(const Date& obj) {
	this->day = obj.day;
	this->month = obj.month;
	this->year = obj.year;
	return *this;
}

int Date::operator-(const Date& obj){
	int countDay = 0;
	countDay = (this->year - obj.year) * 365;
	countDay += (this->month - obj.month) * 31;
	countDay += this->day - obj.day;
	return countDay;
}

int Date::comparisonDate(const Date& obj) {
	if (this->year == obj.year) {
		if (this->month == obj.month) {
			if (this->day == obj.day) {
				return 0;
			} else if (this->day > obj.day) {
				return 1;
			} else if (this->day < obj.day) {
				return -1;
			}
		} else if (this->month > obj.month) {
```

```
                return 1;
            } else if (this->month < obj.month) {
                return -1;
            }
      } else if (this->year > obj.year) {
            return 1;
      } else if (this->year < obj.year) {
            return -1;
      }
      return 0;
}

bool operator<(Date& obj1, Date& obj2) {
      return (obj1.comparisonDate(obj2) < 0) ? true : false;
}

bool operator==(Date& obj1, Date& obj2) {
      return (obj1.comparisonDate(obj2) == 0) ? true : false;
}

istream& operator>>(istream& stream, Date& obj) {
      while(true){
            cout << "День" << endl;
            obj.day = inputShortInt(stream);
            cout << "Месяц" << endl;
            obj.month = inputShortInt(stream);
            cout << "Год" << endl;
            obj.year = inputShortInt(stream);
            if (!(obj.day > 0 && obj.day < 32 && obj.month > 0 && obj.month <
13
                        && obj.year > 1)) {
                  showExceptionScreen();
            }
            else
                  break;
      }
      return stream;
}

ostream& operator<<(ostream& stream, Date& obj) {
      stream << setw(2) << right << obj.day << "." << setw(2) << right
                  << obj.month << "." << setw(4) << right << obj.year;
      return stream;
}

ifstream& operator>>(ifstream& fstream, Date& obj) {
      fstream >> obj.day >> obj.month >> obj.year;
      return fstream;
}

ofstream& operator<<(ofstream& fstream, Date& obj) {
      fstream << obj.day << " " << obj.month << " " << obj.year;
      return fstream;
}
Date Date::getTodayDate(){
      Date todayDate;
      struct tm *tim;
      time_t tt = time(NULL);
      tim = localtime(&tt);
      todayDate.setDay(tim->tm_mday);
      todayDate.setMonth(tim->tm_mon+1);
      todayDate.setYear(tim->tm_year+1900);
      return todayDate;
}
```

Табица 19 – Исходный код в файле Task.h

Task.h

```
#ifndef DATA_CLASSES_TASK_H_
#define DATA_CLASSES_TASK_H_
#include <string>
#include <iostream>
#include <fstream>
#include <ctime>
#include <iomanip>
#include <Windows.h>
#include "Date.h"
using namespace std;

class Task {
private:
	unsigned int taskId;
	string content;
	unsigned int userId;
	Date assignmentDate;
	Date plannedDate;
	Date dateDelivery;
	string notation;
public:
	static const string PATH_TASKS;
	Task();
	Task(unsigned int, string, unsigned int, Date, Date, Date, string);
	Task(const Task&);
	~Task();
	void setTaskId(unsigned int);
	void setContent(string);
	void setUserId(unsigned int);
	void setAssignmentDate(const Date&);
	void setPlannedDate(const Date&);
	void setDateDelivery(const Date&);
	void setNotation(string);
	unsigned int getTaskId();
	string getContent();
	unsigned int getUserId();
	Date& getAssignmentDate();
	Date& getPlannedDate();
	Date& getDateDelivery();
	string getNotation();
	string toString();
	Task& operator=(const Task&);
	friend bool operator<(const Task&, const Task&);
	friend bool operator==(const Task&, const  Task&);
	friend ostream& operator<<(ostream&, Task&);
	friend istream& operator>>(istream&, Task&);
	friend ofstream& operator<<(ofstream&, Task&);
	friend ifstream& operator>>(ifstream&, Task&);

};

#endif /* DATA_CLASSES_TASK_H_ */
```

Табица 20 – Исходный код в файле Task.cpp

Task.cpp

```
#include "Task.h"

const string Task::PATH_TASKS = "./Resources/Tasks.txt";

Task::Task() :
            taskId(0), content(""), userId(0), assignmentDate(),
plannedDate(), dateDelivery(), notation(
                        "") {
}

Task::Task(unsigned int taskId, string content, unsigned int userId,
            Date assignmentDate, Date plannedDate, Date dateDelivery,
            string notation) :
            taskId(taskId), content(content), userId(userId), assignmentDate(
                        assignmentDate), plannedDate(plannedDate),
dateDelivery(
                        dateDelivery), notation(notation) {
}

Task::Task(const Task& obj) :
            taskId(obj.taskId), content(obj.content), userId(obj.userId),
assignmentDate(
                        obj.assignmentDate), plannedDate(obj.plannedDate),
dateDelivery(
                        obj.dateDelivery), notation(obj.notation) {
}

Task::~Task() {
}

void Task::setTaskId(unsigned int taskId) {
      this->taskId = taskId;
}

void Task::setContent(string content) {
      this->content = content;
}

void Task::setUserId(unsigned int userId) {
      this->userId = userId;
}

void Task::setAssignmentDate(const Date& assignmentDate) {
      this->assignmentDate = assignmentDate;
}

void Task::setPlannedDate(const Date& plannedDate) {
      this->plannedDate = plannedDate;
}

void Task::setDateDelivery(const Date& dateDelivery) {
      this->dateDelivery = dateDelivery;
}

void Task::setNotation(string notation) {
      this->notation = notation;
}

unsigned int Task::getTaskId() {
      return this->taskId;
}
```

```
string Task::getContent() {
	return this->content;
}

unsigned int Task::getUserId() {
	return this->userId;
}

Date& Task::getAssignmentDate() {
	return this->assignmentDate;
}

Date& Task::getPlannedDate() {
	return this->plannedDate;
}

Date& Task::getDateDelivery() {
	return this->dateDelivery;
}

string Task::getNotation() {
	return this->notation;
}

string Task::toString() {
	string str;
	str.append("[Task@");
	str.append("taskId=");
	str.append(to_string(this->taskId));
	str.append(",content=");
	str.append(this->content);
	str.append(",userId=");
	str.append(to_string(this->userId));
	str.append(",assignmentDate=");
	str.append(this->assignmentDate.toString());
	str.append(",plannedDate=");
	str.append(this->plannedDate.toString());
	str.append(",dateDelivery=");
	str.append(this->dateDelivery.toString());
	str.append(",notation=");
	str.append(this->notation);
	str.append("]");
	return str;
}

Task& Task::operator=(const Task& obj) {
	this->userId = obj.userId;
	this->content = obj.content;
	this->taskId = obj.taskId;
	this->assignmentDate = obj.assignmentDate;
	this->plannedDate = obj.plannedDate;
	this->dateDelivery = obj.dateDelivery;
	this->notation = obj.notation;
	return *this;
}

bool operator<(const Task& obj1, const  Task& obj2) {
	return obj1.userId < obj2.userId;
}

bool operator==(const Task& obj1, const Task& obj2) {
	return obj1.taskId == obj2.taskId;
}
```

```
ostream& operator<<(ostream& stream, Task& obj) {
	HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
	if(obj.dateDelivery.getYear() == 0){
		if(obj.plannedDate-Date::getTodayDate() < 1){
			SetConsoleTextAttribute(hConsole, (WORD) ( 4 | 0));
		}
		else if(obj.plannedDate-Date::getTodayDate() < 4){
			SetConsoleTextAttribute(hConsole, (WORD) ( 14 | 0));
		}
		else if(obj.plannedDate-Date::getTodayDate() < 7){
			SetConsoleTextAttribute(hConsole, (WORD) ( 1 | 0));
		}
	}
	stream << setw(24) << right << "ID# задания: " << left << obj.taskId <<
"\n"
			<< setw(24) << right << "ID# пользователя: " << left <<
obj.userId << "\n"
			<< setw(24) << right << "Дата выдачи задания: " << left <<
obj.assignmentDate << "\n"
			<< setw(24) << right << "Дата планируемой сдачи: " << left
<< obj.plannedDate << "\n"
			<< setw(24) << right << "Дата сдачи задания: " << left <<
obj.dateDelivery << "\n"
			<< setw(24) << right << "Задание: " << left << obj.content
<< "\n"
			<< setw(24) << right << "Примечание: " << left <<
obj.notation << "\n";
	SetConsoleTextAttribute(hConsole, (WORD) ( 15 | 0));
	return stream;
}
istream& operator>>(istream& stream, Task& obj){
	cout<<"Задания"<<endl;
	obj.content=inputLine(stream);
	cout<<"Идентификатор сотрудника, которому присвоится задание"<<endl;
	obj.userId=inputUnsignedInt(stream);
	struct tm *tim;
	time_t tt = time(NULL);
	tim = localtime(&tt);
	obj.assignmentDate.setDay(tim->tm_mday);
	obj.assignmentDate.setMonth(tim->tm_mon+1);
	obj.assignmentDate.setYear(tim->tm_year+1900);
	cout<<"Дата выдачи задания (сегодняшняя дата): "<< obj.assignmentDate
<<endl;
	cout<<"Планируемая дата сдачи выполненного задания"<<endl;
	stream>>obj.plannedDate;
	cout<<"Примечание"<<endl;
	obj.notation=inputLine(stream);
	return stream;
}
ofstream& operator<<(ofstream& fstream, Task& obj) {
	fstream << obj.taskId << " ";
	fstream << obj.content.length() << " ";
	fstream << obj.content << " ";
	fstream << obj.userId << " ";
	fstream << obj.assignmentDate << " ";
	fstream << obj.plannedDate << " ";
	fstream << obj.dateDelivery << " ";
	fstream << obj.notation.length() << " ";
	fstream << obj.notation;
	return fstream;
}
ifstream& operator>>(ifstream& fstream, Task& obj) {
	size_t size_content, size_notation;
```

```
	fstream >> obj.taskId >> size_content;
	fstream.get();
	obj.content.clear();
	for (size_t i = 0; i < size_content; i++) {
		obj.content += fstream.get();
	}
	fstream >> obj.userId;
	fstream >> obj.assignmentDate;
	fstream >> obj.plannedDate;
	fstream >> obj.dateDelivery;
	fstream >> size_notation;
	fstream.get();
	obj.notation.clear();
	for (size_t i = 0; i < size_notation; i++) {
		obj.notation += fstream.get();
	}
	return fstream;
}
```

Табица 21 – Исходный код в файле UserAccount.h

UserAccount.h

```
#ifndef DATA_CLASSES_USERACCOUNT_H_
#define DATA_CLASSES_USERACCOUNT_H_
#include <string>
#include <iostream>
#include <fstream>
#include <iomanip>
#include "../Converters/HashPasswordWithSal.h"
#include "../Controllers/ControllerIStream.h"
using namespace std;

class UserAccount {
private:
	unsigned int userId;
	string password;
	string surname;
	string name;
	string midname;
	string department;
public:
	static const string PATH_USER_ACCOUNTS;
	UserAccount();
	UserAccount(const UserAccount&);
	UserAccount(unsigned int, string, string, string, string, string);
	~UserAccount();

	void setUserId(const unsigned int);
	void setPasdword(const string password);
	void setSurname(const string surname);
	void setName(const string name);
	void setMidname(const string midname);
	void setDepartment(const string department);
	unsigned int getUserId();
	string getPassword();
	string getSurname();
	string getName();
	string getMidname();
	string getDepartment();
	string toString();
	UserAccount& operator=(const UserAccount&);
	friend bool operator<(const UserAccount&, const UserAccount&);
	friend bool operator==(const UserAccount&, const UserAccount&);
	friend ostream& operator<<(ostream&, UserAccount&);
```

```
      friend istream& operator>>(istream&, UserAccount&);
      friend ofstream& operator<<(ofstream&, UserAccount&);
      friend ifstream& operator>>(ifstream&, UserAccount&);
};

#endif /* DATA_CLASSES_USERACCOUNT_H_ */
```

Табица 22 – Исходный код в файле UserAccount.cpp

UserAccount.cpp

```
#include "UserAccount.h"

const string UserAccount::PATH_USER_ACCOUNTS =
"./Resources/UserAccounts.txt";

UserAccount::UserAccount() :
            userId(0), password(""), surname(""), name(""), midname(""),
department(
                        "") {

}

UserAccount::UserAccount(const UserAccount& obj) :
            userId(obj.userId), password(obj.password), surname(obj.surname),
name(
                        obj.name), midname(obj.midname),
department(obj.department) {

}

UserAccount::UserAccount(unsigned int userId, string password, string
surname,
            string name, string midname, string department) :
            userId(userId), password(password), surname(surname), name(name),
midname(
                        midname), department(department) {

}

UserAccount::~UserAccount() {

}

void UserAccount::setUserId(const unsigned int userId) {
      this->userId = userId;
}

void UserAccount::setPasdword(const string password) {
      this->password = password;
}

void UserAccount::setSurname(const string surname) {
      this->surname = surname;
}

void UserAccount::setName(const string name) {
      this->name = name;
}

void UserAccount::setMidname(const string midname) {
      this->midname = midname;
}
```

```
void UserAccount::setDepartment(const string department) {
	this->department = department;
}

unsigned int UserAccount::getUserId() {
	return userId;
}

string UserAccount::getPassword() {
	return password;
}

string UserAccount::getSurname() {
	return surname;
}

string UserAccount::getName() {
	return name;
}

string UserAccount::getMidname() {
	return midname;
}

string UserAccount::getDepartment() {
	return department;
}

string UserAccount::toString() {
	string str;
	str.append("[UserAccount@");
	str.append("userId=");
	str.append(to_string(getUserId()));
	str.append(",password=");
	str.append(getPassword());
	str.append(",surname=");
	str.append(getSurname());
	str.append(",name=");
	str.append(getName());
	str.append(",midname=");
	str.append(getMidname());
	str.append(",department=");
	str.append(getDepartment());
	str.append("]");
	return str;
}

UserAccount& UserAccount::operator=(const UserAccount& obj) {
	this->userId = obj.userId;
	this->password = obj.password;
	this->surname = obj.surname;
	this->name = obj.name;
	this->midname = obj.midname;
	this->department = obj.department;
	return *this;
}

bool operator<(const UserAccount& obj1, const UserAccount& obj2) {
	return (obj1.surname.compare(obj2.surname) < 0 ? true : false);
}

bool operator==(const UserAccount& obj1, const UserAccount& obj2) {
	return obj1.userId == obj2.userId;
}
```

```
ostream& operator<<(ostream& stream, UserAccount& obj) {
	stream << setw(10) << right << "ID#: " << obj.userId << "\n"
			<< setw(10) << right << "Фамилия: " << obj.surname << "\n"
			<< setw(10) << right << "Имя: " << obj.name << "\n"
			<< setw(10) << right << "Отчество: " << obj.midname << "\n"
			<< setw(10) << right << "Отдел: " << obj.department << "\n";
	return stream;
}

istream& operator>>(istream& stream, UserAccount& obj) {
	cout<<"Пароль"<<endl;
	obj.password=inputString(stream);
	cout<<"Фамилия"<<endl;
	obj.surname=inputString(stream);
	obj.password=HashPasswordWithSal()(obj.password,obj.surname);
	cout<<"Имя"<<endl;
	obj.name=inputString(stream);
	cout<<"Отчество"<<endl;
	obj.midname=inputString(stream);
	cout<<"Отдел"<<endl;
	obj.department=inputString(stream);
	return stream;
}

ofstream& operator<<(ofstream& fstream, UserAccount& obj) {
	fstream << obj.userId << " " << obj.password << " " << obj.surname << "
"
			<< obj.name << " " << obj.midname << " " << obj.department;
	return fstream;
}

ifstream& operator>>(ifstream& fstream, UserAccount& obj) {
	fstream >> obj.userId >> obj.password >> obj.surname >> obj.name
			>> obj.midname >> obj.department;
	return fstream;
}
```

Табица 23 – Исходный код в файле ServiceData.h

ServiceData.h

```
#ifndef SERVICES_SERVICEDATA_H_
#define SERVICES_SERVICEDATA_H_

#include <iostream>
#include <list>
#include "../Data_Classes/UserAccount.h"
#include "../Data_Classes/Task.h"
using namespace std;

template<class T>
void showList(list<T> List) {
	for (T ptr : List) {
		cout << ptr << endl;
	}
}

#endif /* SERVICES_SERVICEDATA_H_ */
```

Табица 24 – Исходный код в файле ServiceTasks.h

ServiceTasks.h

```
#ifndef SERVICES_TASKS_H_
#define SERVICES_TASKS_H_
#include <iostream>
#include <list>
#include <ctime>
#include "../Data_Classes/Task.h"
#include "../Data_Classes/UserAccount.h"
#include "../Controllers/ControllerIOFStream.h"
#include "../Screens/ExceptionScreen.h"
#include "ServiceData.h"

class ServiceTasks {
private:
	ServiceTasks();
	ServiceTasks(const ServiceTasks&);
	~ServiceTasks();
	ServiceTasks& operator=(const ServiceTasks&);
public:
	static const string PATH_FREE_TASKID;

	static bool compTaskByTaskId(Task& first, Task& second);
	static bool compTaskByContent(Task& first, Task& second);
	static bool compTaskByAssignmentDate(Task& first, Task& second);
	static bool compTaskByPlannedDate(Task& first, Task& second);
	static bool compTaskByDateDelivery(Task& first, Task& second);
	static bool compTaskByNotation(Task& first, Task& second);

	static bool recompTaskByTaskId(Task& first, Task& second);
	static bool recompTaskByContent(Task& first, Task& second);
	static bool recompTaskByUserId(Task& first, Task& second);
	static bool recompTaskByAssignmentDate(Task& first, Task& second);
	static bool recompTaskByPlannedDate(Task& first, Task& second);
	static bool recompTaskByDateDelivery(Task& first, Task& second);
	static bool recompTaskByNotation(Task& first, Task& second);

	static list<Task>::iterator findTaskByTaskId(
			list<Task>::iterator begin, list<Task>::iterator end,
			unsigned int taskId);
	static list<Task>::iterator findTaskByContent(
			list<Task>::iterator begin, list<Task>::iterator end,
			string content);
	static list<Task>::iterator findTaskByUserId(
			list<Task>::iterator begin, list<Task>::iterator end,
			unsigned int userId);
	static list<Task>::iterator findTaskByAssignmentDate(
			list<Task>::iterator begin, list<Task>::iterator end,
			Date assignmentDate);
	static list<Task>::iterator findTaskByPlannedDate(
			list<Task>::iterator begin, list<Task>::iterator end,
			Date plannedDate);
	static list<Task>::iterator findTaskByDateDelivery(
			list<Task>::iterator begin, list<Task>::iterator end,
			Date dateDelivery);
	static list<Task>::iterator findTaskByNotation(
			list<Task>::iterator begin, list<Task>::iterator end,
			string notation);

	static void addTask(list<Task>&, list<UserAccount>&, list<unsigned
int>&);
	static void editingContentTask(list<Task>::iterator&);
	static void editingUserIdTask(list<Task>::iterator&,
```

```
list<UserAccount>&);
      static void editingPlannedDateTask(list<Task>::iterator&);
      static void editingDateDeliveryTask(list<UserAccount>::iterator&,
list<Task>::iterator&);
      static void editingNotationTask(list<Task>::iterator&);
      static void editingAllDataTask(list<Task>::iterator&,
list<UserAccount>&);

      static void deleteTask(list<Task>&, list<unsigned int>&);
      static void deleteTaskByIterator(list<Task>&, list<Task>::iterator&,
list<unsigned int>&);
      static void deleteAllTask(list<Task>&, list<unsigned int>&);

      static void showOverdueTasks(list<Task>&);
      static void showTasksByUserId(list<Task>&);
      static void showOverdueTasksToThreeDays(list<Task>&);
};

#endif /* SERVICES_TASKS_H_ */
```

Табица 25 – Исходный код в файле ServiceTasks.cpp

ServiceTasks.cpp

```
#include "ServiceTasks.h"

const string ServiceTasks::PATH_FREE_TASKID = "./Resources/FreeTaskId.txt";

ServiceTasks::ServiceTasks() {
}

ServiceTasks::~ServiceTasks() {
}

bool ServiceTasks::compTaskByTaskId(Task& first, Task& second) {
      return first.getTaskId() < second.getTaskId();
}
bool ServiceTasks::compTaskByContent(Task& first, Task& second) {
      return (first.getContent().compare(second.getContent()) == -1 ? true :
false);
}
bool ServiceTasks::compTaskByAssignmentDate(Task& first, Task& second) {
      return
(first.getAssignmentDate().comparisonDate(second.getAssignmentDate())
                  == -1 ? true : false);
}
bool ServiceTasks::compTaskByPlannedDate(Task& first, Task& second) {
      return (first.getPlannedDate().comparisonDate(second.getPlannedDate())
                  == -1 ? true : false);
}
bool ServiceTasks::compTaskByDateDelivery(Task& first, Task& second) {
      return (first.getDateDelivery().comparisonDate(second.getDateDelivery())
                  == -1 ? true : false);
}
bool ServiceTasks::compTaskByNotation(Task& first, Task& second) {
      return (first.getNotation().compare(second.getNotation()) == -1 ?
                  true : false);
}

bool ServiceTasks::recompTaskByTaskId(Task& first, Task& second){
      return first.getTaskId() > second.getTaskId();
}
bool ServiceTasks::recompTaskByContent(Task& first, Task& second){
      return (first.getContent().compare(second.getContent()) == 1 ? true :
```

```
false);
}
bool ServiceTasks::recompTaskByUserId(Task& first, Task& second){
      return first.getUserId() > second.getUserId();
}
bool ServiceTasks::recompTaskByAssignmentDate(Task& first, Task& second){
      return
(first.getAssignmentDate().comparisonDate(second.getAssignmentDate())
                  == 1 ? true : false);
}
bool ServiceTasks::recompTaskByPlannedDate(Task& first, Task& second){
      return (first.getPlannedDate().comparisonDate(second.getPlannedDate())
                  == 1 ? true : false);
}
bool ServiceTasks::recompTaskByDateDelivery(Task& first, Task& second){
      return (first.getDateDelivery().comparisonDate(second.getDateDelivery())
                  == 1 ? true : false);
}
bool ServiceTasks::recompTaskByNotation(Task& first, Task& second){
      return (first.getNotation().compare(second.getNotation()) == 1 ?
                        true : false);
}

list<Task>::iterator ServiceTasks::findTaskByTaskId(
            list<Task>::iterator begin, list<Task>::iterator end,
            unsigned int taskId){
      for (; begin != end; begin++)
            if
(to_string((*begin).getTaskId()).find(to_string(taskId))!=string::npos)
                  return begin;
      return end;
}
list<Task>::iterator ServiceTasks::findTaskByContent(
            list<Task>::iterator begin, list<Task>::iterator end,
            string content){
      for (; begin != end; begin++)
            if ((*begin).getContent().find(content)!=string::npos)
                  return begin;
      return end;
}
list<Task>::iterator ServiceTasks::findTaskByUserId(
            list<Task>::iterator begin, list<Task>::iterator end,
            unsigned int userId){
      for (; begin != end; begin++)
            if
(to_string((*begin).getUserId()).find(to_string(userId))!=string::npos)
                  return begin;
      return end;
}
list<Task>::iterator ServiceTasks::findTaskByAssignmentDate(
            list<Task>::iterator begin, list<Task>::iterator end,
            Date assignmentDate){
      for (; begin != end; begin++)
            if
((*begin).getAssignmentDate().comparisonDate(assignmentDate)==0)
                  return begin;
      return end;
}
list<Task>::iterator ServiceTasks::findTaskByPlannedDate(
            list<Task>::iterator begin, list<Task>::iterator end,
            Date plannedDate){
      for (; begin != end; begin++)
            if ((*begin).getPlannedDate().comparisonDate(plannedDate)==0)
                  return begin;
```

```
	return end;
}
list<Task>::iterator ServiceTasks::findTaskByDateDelivery(
		list<Task>::iterator begin, list<Task>::iterator end,
		Date dateDelivery){
	for (; begin != end; begin++)
		if ((*begin).getDateDelivery().comparisonDate(dateDelivery)==0)
			return begin;
	return end;
}
list<Task>::iterator ServiceTasks::findTaskByNotation(
		list<Task>::iterator begin, list<Task>::iterator end,
		string notation){
	for (; begin != end; begin++)
		if ((*begin).getNotation().find(notation)!=string::npos)
			return begin;
	return end;
}

void ServiceTasks::addTask(list<Task> &ListTasks, list<UserAccount>
&ListUserAccounts, list<unsigned int> &ListFreeTaskId){
	system("cls");
	unsigned int maxTaskIdInListNow = (*ListTasks.begin()).getUserId();
	if (ListFreeTaskId.size() == 0) {
		for (Task t : ListTasks) {
			if (maxTaskIdInListNow < t.getTaskId())
				maxTaskIdInListNow = t.getTaskId();
		}
		maxTaskIdInListNow++;
	} else {
		maxTaskIdInListNow = *ListFreeTaskId.begin();
		ListFreeTaskId.pop_front();
	}
	Task newTask;
	newTask.setTaskId(maxTaskIdInListNow);
	cout << "Введите данные о новом задании с идентификатором "
			<< newTask.getTaskId() << "\n" << endl;
	cin >> newTask;
	if(newTask.getAssignmentDate().comparisonDate(newTask.getPlannedDate())
< 1){
		bool FalseInputUserId = false;
		for(UserAccount ua:ListUserAccounts)
			if(newTask.getUserId()==ua.getUserId()){
				FalseInputUserId = true;
				cout << "Подтвердите создание нового задания (да/нет)"
						<< endl;
				cout << newTask << endl;
				string confirmation(inputString(cin));
				if (confirmation == "да")
					ListTasks.push_back(newTask);
				if (confirmation != "да" && confirmation != "нет")
					showExceptionScreen();
			}
		if(!FalseInputUserId)
			showNotFindUserId(newTask.getUserId());
	}
	else
		showNotCompDate();
}

void ServiceTasks::editingContentTask(list<Task>::iterator &findTask){
	cout<<"Задание"<<endl;
	(*findTask).setContent(inputLine(cin));
}
```

```
void ServiceTasks::editingUserIdTask(list<Task>::iterator &findTask,
list<UserAccount> &ListUserAccounts){
	cout<<"Идентификатор пользователя, которому будет пренадлежать
задание"<<endl;
	unsigned int valueUserId = inputUnsignedInt(cin);
	bool FalseInputUserId = false;
	for(UserAccount ua:ListUserAccounts)
		if(ua.getUserId() == valueUserId){
			(*findTask).setUserId(valueUserId);
			FalseInputUserId = true;
			break;
		}
	if(!FalseInputUserId)
		showNotFindUserId(valueUserId);
}

void ServiceTasks::editingPlannedDateTask(list<Task>::iterator &findTask){
	cout<<"Планируемая дата сдачи заданя"<<endl;
	Date valuePlannedDate;
	cin >> valuePlannedDate;
	if((*findTask).getAssignmentDate().comparisonDate(valuePlannedDate) < 1)
		(*findTask).setPlannedDate(valuePlannedDate);
	else
		showNotCompDate();
}

void ServiceTasks::editingDateDeliveryTask(list<UserAccount>::iterator
&activeAccount,list<Task>::iterator &findTask){
	if((*activeAccount).getUserId() !=
0&&(*findTask).getPlannedDate().comparisonDate(Date::getTodayDate()) < 1){
		showNotPlannedDate();
	}
	else{
		cout << "Дата сдачи задания ";
		Date todayDate(Date::getTodayDate());
		cout << todayDate << endl;
		(*findTask).setDateDelivery(Date::getTodayDate());
		system("pause");
	}
}

void ServiceTasks::editingNotationTask(list<Task>::iterator &findTask){
	cout << "Примечане" <<endl;
	(*findTask).setNotation(inputLine(cin));
}

void ServiceTasks::editingAllDataTask(list<Task>::iterator &findTask,
list<UserAccount> &ListUserAccounts){
	Task valueInputTask;
	valueInputTask.setTaskId((*findTask).getTaskId());
	cout<<"Задания"<<endl;
	valueInputTask.setContent(inputLine(cin));
	cout<<"Идентификатор пользователя, которому будет пренадлежать
задание"<<endl;
	valueInputTask.setUserId(inputUnsignedInt(cin));
	bool FalseInputUserId = false;
	for(UserAccount ua:ListUserAccounts)
		if(ua.getUserId() == valueInputTask.getUserId()){
			FalseInputUserId = true;
			break;
		}
	if(!FalseInputUserId)
		showNotFindUserId(valueInputTask.getUserId());
```

```
	if(FalseInputUserId){
		valueInputTask.setAssignmentDate((*findTask).getAssignmentDate());
		cout<<"Планируемая дата сдачи заданя"<<endl;
		Date valuePlannedDate;
		cin >> valuePlannedDate;

	if((*findTask).getAssignmentDate().comparisonDate(valuePlannedDate) <
1){
			valueInputTask.setPlannedDate(valuePlannedDate);
			cout << valueInputTask.getPlannedDate() << endl;
			cout<<"Примечание"<<endl;
			valueInputTask.setNotation(inputLine(cin));
			(*findTask) = valueInputTask;
		}
		else
			showNotCompDate();
	}
}

void ServiceTasks::deleteTask(list<Task> &ListTasks, list<unsigned int>
&ListFreeTaskId){
	system("cls");
	cout
			<< "ВВедите идентификатор  задания, данные которого
необходимо удалить"
			<< endl;
	unsigned int inputTaskId = inputUnsignedInt(cin);
	ListTasks.sort(ServiceTasks::compTaskByTaskId);
	list<Task>::iterator findTask =
ServiceTasks::findTaskByTaskId(ListTasks.begin(),ListTasks.end(),inputTaskId)
;
	if (findTask != ListTasks.end()) {
		cout
				<< "Подтвердите удаление учетной записи пользователя и
всех его заданий (да/нет)"
				<< endl;
		cout << (*findTask) << endl;
		string confirmation(inputString(cin));
		if (confirmation == "да") {
			ServiceTasks::deleteTaskByIterator(ListTasks, findTask,
ListFreeTaskId);
		}
		if (confirmation != "да" && confirmation != "нет")
			showExceptionScreen();
	}
	else{
		showNotFindData();
	}

}

void ServiceTasks::deleteTaskByIterator(list<Task> &ListTasks,
list<Task>::iterator &findTask, list<unsigned int> &ListFreeTaskId){
	ListFreeTaskId.push_back((*findTask).getTaskId());
	ListTasks.remove((*findTask));
}

void ServiceTasks::deleteAllTask(list<Task> &ListTasks, list<unsigned int>
&ListFreeTaskId){
	cout
			<< "Подтвердите удаление всех учетных записей пользователей
и всех заданий (да/нет)"
			<< endl;
	string confirmation(inputString(cin));
```

```
	if (confirmation == "да") {
		while(true){
			list<Task>::iterator begin = ListTasks.begin();
			list<Task>::iterator end = ListTasks.end();
			if(begin == end)
				break;
			ServiceTasks::deleteTaskByIterator(ListTasks, begin,
ListFreeTaskId);
		}
	}
	if (confirmation != "да" && confirmation != "нет")
		showExceptionScreen();
}

void ServiceTasks::showOverdueTasks(list<Task> &ListTasks){
	if(ListTasks.size()!=0){
		ListTasks.sort(ServiceTasks::compTaskByTaskId);
		system("cls");
		list<Task> findListTasks;
		list<Task>::iterator begin = ListTasks.begin();
		list<Task>::iterator end = ListTasks.end();
		for(Task t:ListTasks)
			if(t.getDateDelivery().getYear()==0&&t.getPlannedDate()-
Date::getTodayDate()<1)
				findListTasks.push_back(t);
		if(findListTasks.size() != 0){
			showList(findListTasks);
			system("pause");
		}
		else
			showNotFindData();
	}
	else
		showNotData();
}

void ServiceTasks::showTasksByUserId(list<Task> &ListTasks){
	if(ListTasks.size()!=0){
		ListTasks.sort();
		system("cls");
		list<Task> findListTasks;
		list<Task>::iterator begin = ListTasks.begin();
		list<Task>::iterator end = ListTasks.end();
		unsigned int userId;
		cout
			<< "Введите идентификатор для поиска нужной записи"
			<< endl;
		userId = inputUnsignedInt(cin);
		while (true) {
			begin = ServiceTasks::findTaskByUserId(begin,end,userId);
			if (begin == end)
				break;
			findListTasks.push_back(*begin);
			begin++;
		}
		if (findListTasks.size() != 0) {
			showList(findListTasks);
			system("pause");
		} else
			showNotFindData();
	}
	else
		showNotData();
}
```

```
void ServiceTasks::showOverdueTasksToThreeDays(list<Task> &ListTasks){
	if(ListTasks.size()!=0){
		ListTasks.sort(ServiceTasks::compTaskByTaskId);
		system("cls");
		list<Task> findListTasks;
		list<Task>::iterator begin = ListTasks.begin();
		list<Task>::iterator end = ListTasks.end();
		for(Task t:ListTasks)
			if(t.getDateDelivery().getYear()==0&&t.getPlannedDate()-
Date::getTodayDate()>0&&t.getPlannedDate()-Date::getTodayDate()<4)
				findListTasks.push_back(t);
		if(findListTasks.size() != 0){
			showList(findListTasks);
			system("pause");
		}
		else
			showNotFindData();
	}
	else
		showNotData();
}
```

Табица 26 – Исходный код в файле ServiceUserAccounts.h

ServiceUserAccounts.h

```
#ifndef SERVICES_USERACCOUNTS_H_
#define SERVICES_USERACCOUNTS_H_
#include <iostream>
#include <list>
#include "../Data_Classes/UserAccount.h"
#include "../Controllers/ControllerIOFStream.h"
#include "../Screens/ExceptionScreen.h"
#include "../Data_Classes/Task.h"
#include "ServiceTasks.h"
using namespace std;

class ServiceUserAccounts {
private:
	ServiceUserAccounts();
	ServiceUserAccounts(const ServiceUserAccounts&);
	~ServiceUserAccounts();
	ServiceUserAccounts& operator=(const ServiceUserAccounts&);
public:
	static const string PATH_FREE_USERID;

	static bool compUserAccountByUserId(UserAccount& first, UserAccount&
second);
	static bool compUserAccountByName(UserAccount& first, UserAccount&
second);
	static bool compUserAccountByMidname(UserAccount& first, UserAccount&
second);
	static bool compUserAccountByDepartment(UserAccount& first, UserAccount&
second);
	static bool recompUserAccountByUserId(UserAccount& first, UserAccount&
second);
	static bool recompUserAccountBySurname(UserAccount& first, UserAccount&
second);
	static bool recompUserAccountByName(UserAccount& first, UserAccount&
second);
	static bool recompUserAccountByMidname(UserAccount& first, UserAccount&
second);
	static bool recompUserAccountByDepartment(UserAccount& first,
```

```
UserAccount& second);


      static list<UserAccount>::iterator findUserAccountByUserId(
                  list<UserAccount>::iterator begin,
list<UserAccount>::iterator end,
                  unsigned int userId);
      static list<UserAccount>::iterator findUserAccountByPassword(
                  list<UserAccount>::iterator begin,
list<UserAccount>::iterator end,
                  string password);
      static list<UserAccount>::iterator findUserAccountBySurname(
                  list<UserAccount>::iterator begin,
list<UserAccount>::iterator end,
                  string surname);
      static list<UserAccount>::iterator findUserAccountByName(
                  list<UserAccount>::iterator begin,
list<UserAccount>::iterator end,
                  string name);
      static list<UserAccount>::iterator findUserAccountByMidname(
                  list<UserAccount>::iterator begin,
list<UserAccount>::iterator end,
                  string midname);
      static list<UserAccount>::iterator findUserAccountByDepartment(
                  list<UserAccount>::iterator begin,
list<UserAccount>::iterator end,
                  string department);

      static void addUserAccount(list<UserAccount>&, list<unsigned int>&);
      static void editingSNMUserAccount(list<UserAccount>::iterator&, string);
      static void editingDepartmentUserAccount(list<UserAccount>::iterator&);
      static void editingPasswordUserAccount(list<UserAccount>::iterator&,
string);
      static void editingAllDataUserAccount(list<UserAccount>::iterator&);

      static void deleteUserAccountAndTasks(list<UserAccount>&, list<Task>&,
list<unsigned int>&, list<unsigned int>&);
      static void deleteUserAccountAndTasksByIterator(list<UserAccount>&,
list<Task>&, list<UserAccount>::iterator&, list<unsigned int>&, list<unsigned
int>&);
      static void deleteAllUserAccountsAndTasks(list<UserAccount>&,
list<Task>&, list<unsigned int>&, list<unsigned int>&);
};

#endif /* SERVICES_USERACCOUNTS_H_ */
```

Табица 27 – Исходный код в файле ServiceUserAccounts.cpp

ServiceUserAccounts.cpp

```
#include "ServiceUserAccounts.h"

const string ServiceUserAccounts::PATH_FREE_USERID =
            "./Resources/FreeUserId.txt";

ServiceUserAccounts::ServiceUserAccounts() {
}

ServiceUserAccounts::~ServiceUserAccounts() {
}

bool ServiceUserAccounts::compUserAccountByUserId(UserAccount& first,
           UserAccount& second) {
      return first.getUserId() < second.getUserId();
```

```
}
bool ServiceUserAccounts::compUserAccountByName(UserAccount& first,
            UserAccount& second) {
      return (first.getName().compare(second.getName()) == -1 ? true : false);
}
bool ServiceUserAccounts::compUserAccountByMidname(UserAccount& first,
            UserAccount& second) {
      return (first.getMidname().compare(second.getMidname()) == -1 ? true :
false);
}
bool ServiceUserAccounts::compUserAccountByDepartment(UserAccount& first,
            UserAccount& second) {
      return (first.getDepartment().compare(second.getDepartment()) == -1 ?
                  true : false);
}
bool ServiceUserAccounts::recompUserAccountByUserId(UserAccount& first,
            UserAccount& second) {
      return first.getUserId() > second.getUserId();
}
bool ServiceUserAccounts::recompUserAccountBySurname(UserAccount& first,
            UserAccount& second) {
      return (first.getSurname().compare(second.getSurname()) == 1 ? true :
false);
}
bool ServiceUserAccounts::recompUserAccountByName(UserAccount& first,
            UserAccount& second) {
      return (first.getName().compare(second.getName()) == 1 ? true : false);
}
bool ServiceUserAccounts::recompUserAccountByMidname(UserAccount& first,
            UserAccount& second) {
      return (first.getMidname().compare(second.getMidname()) == 1 ? true :
false);
}
bool ServiceUserAccounts::recompUserAccountByDepartment(UserAccount& first,
            UserAccount& second) {
      return (first.getDepartment().compare(second.getDepartment()) == 1 ?
                  true : false);
}

list<UserAccount>::iterator ServiceUserAccounts::findUserAccountByUserId(
            list<UserAccount>::iterator begin, list<UserAccount>::iterator
end,
            unsigned int userId) {
      for (; begin != end; begin++)
            if
(to_string((*begin).getUserId()).find(to_string(userId))!=string::npos)
                  return begin;
      return end;
}

list<UserAccount>::iterator ServiceUserAccounts::findUserAccountByPassword(
            list<UserAccount>::iterator begin, list<UserAccount>::iterator
end,
            string password) {
      for (; begin != end; begin++)
            if ((*begin).getPassword().find(password) != string::npos)
                  return begin;
      return end;
}

list<UserAccount>::iterator ServiceUserAccounts::findUserAccountBySurname(
            list<UserAccount>::iterator begin, list<UserAccount>::iterator
end,
            string surname) {
```

```
	for (; begin != end; begin++)
		if ((*begin).getSurname().find(surname) != string::npos)
			return begin;
	return end;
}

list<UserAccount>::iterator ServiceUserAccounts::findUserAccountByName(
		list<UserAccount>::iterator begin, list<UserAccount>::iterator
end,
		string name) {
	for (; begin != end; begin++)
		if ((*begin).getName().find(name) != string::npos)
			return begin;
	return end;
}

list<UserAccount>::iterator ServiceUserAccounts::findUserAccountByMidname(
		list<UserAccount>::iterator begin, list<UserAccount>::iterator
end,
		string midname) {
	for (; begin != end; begin++)
		if ((*begin).getMidname().find(midname) != string::npos)
			return begin;
	return end;
}

list<UserAccount>::iterator ServiceUserAccounts::findUserAccountByDepartment(
		list<UserAccount>::iterator begin, list<UserAccount>::iterator
end,
		string department) {
	for (; begin != end; begin++)
		if ((*begin).getDepartment().find(department) != string::npos)
			return begin;
	return end;
}

void ServiceUserAccounts::addUserAccount(list<UserAccount> &ListUserAccounts,
list<unsigned int> &ListFreeUserId) {
	system("cls");
	unsigned int maxUserIdInListNow =
(*ListUserAccounts.begin()).getUserId();
	if (ListFreeUserId.size() == 0) {
		for (UserAccount ua : ListUserAccounts) {
			if (maxUserIdInListNow < ua.getUserId())
				maxUserIdInListNow = ua.getUserId();
		}
		maxUserIdInListNow++;
	} else {
		maxUserIdInListNow = *ListFreeUserId.begin();
		ListFreeUserId.pop_front();
	}
	UserAccount newUser;
	newUser.setUserId(maxUserIdInListNow);
	cout << "Введите данные о новом пользователе с идентификатором "
			<< newUser.getUserId() << "\n" << endl;
	cin >> newUser;
	cout << "Подтвердите создание новой учетной записи пользователя
(да/нет)"
			<< endl;
	cout << newUser << endl;
	string confirmation(inputString(cin));
	if (confirmation == "да")
		ListUserAccounts.push_back(newUser);
	if (confirmation != "да" && confirmation != "нет")
```

```
            showExceptionScreen();
}

void ServiceUserAccounts::editingSNMUserAccount(list<UserAccount>::iterator&
it,
            string passwordAccount) {
      cout << "Фамилия" << endl;
      (*it).setSurname(inputString(cin));
      (*it).setPasdword(
                  HashPasswordWithSal()(passwordAccount, (*it).getSurname()));
      cout << "Имя" << endl;
      (*it).setName(inputString(cin));
      cout << "Отчество" << endl;
      (*it).setMidname(inputString(cin));
}

void ServiceUserAccounts::editingDepartmentUserAccount(
            list<UserAccount>::iterator& it) {
      cout << "Отдел" << endl;
      (*it).setDepartment(inputString(cin));
}

void ServiceUserAccounts::editingPasswordUserAccount(
            list<UserAccount>::iterator& it, string passwordAccount) {
      cout << "Новый пароль" << endl;
      passwordAccount = inputString(cin);
      (*it).setPasdword(
                  HashPasswordWithSal()(passwordAccount, (*it).getSurname()));
}

void ServiceUserAccounts::editingAllDataUserAccount(
            list<UserAccount>::iterator& it) {
      cin >> (*it);
}

void ServiceUserAccounts::deleteUserAccountAndTasks(
            list<UserAccount> &ListUserAccounts, list<Task> &ListTasks,
list<unsigned int> &ListFreeUserId, list<unsigned int> &ListFreeTaskId) {
      system("cls");
      cout
                  << "ВВедите индентификатор пользователя, данные которого
необходимо удалить"
                  << endl;
      unsigned int inputUserId = inputUnsignedInt(cin);
      ListUserAccounts.sort(ServiceUserAccounts::compUserAccountByUserId);
      ListTasks.sort(ServiceTasks::compTaskByTaskId);
      if (inputUserId != 0) {
            list<UserAccount>::iterator findUserAccount =
ServiceUserAccounts::findUserAccountByUserId(
                        ListUserAccounts.begin(), ListUserAccounts.end(),
inputUserId);
            if (findUserAccount != ListUserAccounts.end()) {
                  cout
                              << "Подтвердите удаление учетной записи
пользователя и всех его заданий (да/нет)"
                              << endl;
                  cout << (*findUserAccount) << endl;
                  string confirmation(inputString(cin));
                  if (confirmation == "да")
                        deleteUserAccountAndTasksByIterator(ListUserAccounts,
ListTasks, findUserAccount, ListFreeUserId, ListFreeTaskId);
                  if (confirmation != "да" && confirmation != "нет")
                        showExceptionScreen();
            } else
```

```
                  showNotFindData();
      } else
            cout << "Администратора удалить нельзя" << endl;
}

void
ServiceUserAccounts::deleteUserAccountAndTasksByIterator(list<UserAccount>
&ListUserAccounts, list<Task> &ListTasks, list<UserAccount>::iterator
&findUserAccount, list<unsigned int> &ListFreeUserId, list<unsigned int>
&ListFreeTaskId){
      while (true) {
            list<Task>::iterator begin = ListTasks.begin();
            list<Task>::iterator end = ListTasks.end();
            begin = ServiceTasks::findTaskByUserId(begin, end,
                        (*findUserAccount).getUserId());
            if (begin == end)
                  break;
            ServiceTasks::deleteTaskByIterator(ListTasks, begin,
ListFreeTaskId);
      }
      ListFreeUserId.push_back((*findUserAccount).getUserId());
      ListUserAccounts.remove((*findUserAccount));
}

void ServiceUserAccounts::deleteAllUserAccountsAndTasks(
            list<UserAccount> &ListUserAccounts, list<Task> &ListTasks,
list<unsigned int> &ListFreeUserId, list<unsigned int> &ListFreeTaskId) {
      cout
                  << "Подтвердите удаление всех учетных записей пользователей
и всех заданий (да/нет)"
                  << endl;
      string confirmation(inputString(cin));
      if (confirmation == "да") {
            while(true){
                  list<UserAccount>::iterator begin =
++ListUserAccounts.begin();
                  list<UserAccount>::iterator end = ListUserAccounts.end();
                  if(begin == end)
                        break;

      ServiceUserAccounts::deleteUserAccountAndTasksByIterator(ListUserAccount
s, ListTasks, begin, ListFreeUserId, ListFreeTaskId);
            }
      }
      if (confirmation != "да" && confirmation != "нет")
            showExceptionScreen();

}
```

Табица 28 – Исходный код в файле LoginAndPasswordScreen.h

LoginAndPasswordScreen.h

```
#ifndef SCREENS_LOGINANDPASSWORDSCREEN_H_
#define SCREENS_LOGINANDPASSWORDSCREEN_H_
#include <iostream>
#include <windows.h>
#include <list>

#include "../Converters/HashPasswordWithSal.h"
#include "../Services/ServiceData.h"
#include "../Services/ServiceUserAccounts.h"
#include "../Data_Classes/UserAccount.h"
#include "../Controllers/ControllerIStream.h"
#include "ExceptionScreen.h"
using namespace std;
```

```
class LoginAndPasswordScreen{
private:
	list<UserAccount> *ListUserAccounts;
	static bool created;

	LoginAndPasswordScreen(list<UserAccount>&);
	LoginAndPasswordScreen(const LoginAndPasswordScreen&);
	~LoginAndPasswordScreen();
	LoginAndPasswordScreen& operator=(const LoginAndPasswordScreen&);
	list<UserAccount>::iterator inputAndCheckLoginAndPassword();
public:
	static LoginAndPasswordScreen& instance(list<UserAccount>&);
	list<UserAccount>::iterator showLoginAndPasswordScreen();
};

#endif /* SCREENS_LOGINANDPASSWORDSCREEN_H_ */
```

Табица 29 – Исходный код в файле LoginAndPasswordScreen.cpp

LoginAndPasswordScreen.cpp

```
#include "LoginAndPasswordScreen.h"

bool LoginAndPasswordScreen::created=false;

LoginAndPasswordScreen::LoginAndPasswordScreen(list<UserAccount>
&ListUserAccounts){
	this->ListUserAccounts=&ListUserAccounts;
}

LoginAndPasswordScreen::~LoginAndPasswordScreen(){
	delete ListUserAccounts;
}

LoginAndPasswordScreen& LoginAndPasswordScreen::instance(list<UserAccount>
&ListUserAccounts){
	static LoginAndPasswordScreen *inst;
	if (!created)
		inst = new LoginAndPasswordScreen(ListUserAccounts);
	return *inst;
}

list<UserAccount>::iterator
LoginAndPasswordScreen::showLoginAndPasswordScreen() {
	list<UserAccount>::iterator activeAccount;
	while(true){
		activeAccount = inputAndCheckLoginAndPassword();
		if (activeAccount != ListUserAccounts->end()){
			system("cls");
			cout<<"Вы вошли под учетной записью"<<endl;
			cout << (*activeAccount) << endl;
			system("pause");
			break;
		}
	}
	return activeAccount;
}

list<UserAccount>::iterator
LoginAndPasswordScreen::inputAndCheckLoginAndPassword(){
	system("cls");
	cout << "----------Форма входа в систему----------" <<endl;
	cout << "Введите логин и пароль для входа в систему" << endl;
```

```
      cout << "Логин" << endl;
      unsigned int login = inputUnsignedInt(cin);
      cout << "Пароль" << endl;
      string password = inputString(cin);
      list<UserAccount>::iterator it =
ServiceUserAccounts::findUserAccountByUserId(ListUserAccounts->begin(),
                  ListUserAccounts->end(), login);
      if (it == ListUserAccounts->end()){
            showNotFindData();
            return ListUserAccounts->end();
      }
      if ((*it).getPassword().compare(
                  HashPasswordWithSal()(password, (*it).getSurname())) != 0){
            showExceptionScreen();
            return ListUserAccounts->end();
      }
      return it;
}
```

Табица 30 – Исходный код в файле ExceptionScreen.h

ExceptionScreen.h

```
#ifndef SCREENS_EXCEPTIONSCREEN_H_
#define SCREENS_EXCEPTIONSCREEN_H_
#include <iostream>
#include <cstdlib>
#include "../Data_Classes/Date.h"
using namespace std;


void showExceptionScreen();
void showNotData();
void showNotFindData();
void showNotFindUserId(unsigned int);
void showNotCompDate();
void showNotPlannedDate();

#endif /* SCREENS_EXCEPTIONSCREEN_H_ */
```

Табица 31 – Исходный код в файле ExceptionScreen.cpp

ExceptionScreen.cpp

```
#include "ExceptionScreen.h"

void showExceptionScreen(){
      system("cls");
      cout<<"ОШИБКА ВВОДА ДАННЫХ! ПОПРОБУЙТЕ ЕЩЕ РАЗ!"<<endl;
      system("pause");
}

void showNotFindData(){
      system("cls");
      cout<<"ПО ВАШЕМУ ЗАПРОСУ ДАННЫХ НЕ СУЩЕСТВУЕТ! ПОПРОБУЙТЕ ЕЩЕ
РАЗ!"<<endl;
      system("pause");
}

void showNotData(){
      system("cls");
      cout<<"ДАННЫХ НЕТ В БАЗЕ ДАННЫХ! НЕТ ВОЗМОЖНОСТИ МАНИПУЛЯЦИЙ!"<<endl;
      system("pause");
}
```

```
void showNotFindUserId(unsigned int UserId){
	system("cls");
	cout<<"ПОЛЬЗОВАТЕЛЯ С ТАКИМ ID = " << UserId << " НЕТ В БАЗЕ
ДАННЫХ!"<<endl;
	system("pause");
}

void showNotCompDate(){
	system("cls");
	cout<<"ПЛАНИРУЕМАЯ ДАТА СДАЧИ НЕ МОЖЕТ БЫТЬ МЕНЬШЕ ДАТЫ ВЫДАЧИ ЗАДАНИЯ"
<<endl;
	system("pause");
}

void showNotPlannedDate(){
	system("cls");
	cout << "ВРЕМЯ СДАЧИ ЗАДАНИЯ ПРОСРОЧЕНО ПО ЭТОЙ ПРИЧИНЕ ОБРАТИТЕСЬ К
АДМИНИСТРАТОРУ" <<endl;
	system("pause");
}
```

Табица 32 – Исходный код в файле AdministratorScreen.h

AdministratorScreen.h

```
#ifndef SCREENS_ADMINISTRATORSCREEN_H_
#define SCREENS_ADMINISTRATORSCREEN_H_

#include <iostream>
#include <string>
#include <list>
#include "../Data_Classes/UserAccount.h"
#include "../Data_Classes/Task.h"
#include "../Services/ServiceData.h"
#include "../Services/ServiceUserAccounts.h"
#include "../Services/ServiceTasks.h"
#include "../Controllers/ControllerIStream.h"
#include "../Controllers/ControllerIOFStream.h"
#include "ExceptionScreen.h"
using namespace std;

class AdministratorScreen {
private:
	list<UserAccount> *ListUserAccounts;
	list<unsigned int> ListFreeUserId;
	list<Task> *ListTasks;
	list<unsigned int> ListFreeTaskId;
	static bool created;

	AdministratorScreen(list<UserAccount>&, list<Task>&);
	AdministratorScreen(const AdministratorScreen&);
	~AdministratorScreen();
	AdministratorScreen& operator=(const AdministratorScreen&);
	void accountManagementMenu();
	void tasksAndStatisticsMenu();
	void editingUserAccountMenu();
	void editingTaskMenu();
	void sortInAscendingOrderUserAccountsMenu();
	void sortInAscendingOrderTasksMenu();
	void sortInDecreasingOrderUserAccountsMenu();
	void sortInDecreasingOrderTasksMenu();
	void findUserAccountMenu();
	void findTaskMenu();
public:
```

```
      static AdministratorScreen& instance(list<UserAccount>&, list<Task>&);
      void showAdministratorScreen();
};

#endif /* SCREENS_ADMINISTRATORSCREEN_H_ */
```

Табица 33 – Исходный код в файле AdministratorScreen.cpp

AdministratorScreen.cpp

```
#include "AdministratorScreen.h"

bool AdministratorScreen::created = false;
AdministratorScreen::AdministratorScreen(list<UserAccount> &ListUserAccounts,
            list<Task> &ListTasks) {
      this->ListUserAccounts = &ListUserAccounts;
      this->ListTasks = &ListTasks;
      this->created = true;
      this->ListFreeUserId = readFile(ServiceUserAccounts::PATH_FREE_USERID,
                  ListFreeUserId);
      this->ListFreeTaskId = readFile(ServiceTasks::PATH_FREE_TASKID,
                  ListFreeTaskId);
}

AdministratorScreen::~AdministratorScreen() {
}

AdministratorScreen& AdministratorScreen::instance(
            list<UserAccount> &ListUserAccounts, list<Task> &ListTasks) {
      static AdministratorScreen *inst;
      if (!created)
            inst = new AdministratorScreen(ListUserAccounts, ListTasks);
      return *inst;
}

void AdministratorScreen::showAdministratorScreen() {
      unsigned int variation;
      while (true) {
            system("cls");
            cout<<"----------Главное меню администратора----------"<<endl;
            cout << "1.\tУчетные записи пользователей\n"
                        << "2.\tЗадания и статистика\n" << "3.\tСохранение
изменений\n"
                        << "0.\tВыйти" << endl;
            variation = inputUnsignedInt(cin);
            switch (variation) {
            case 1:
                  accountManagementMenu();
                  break;
            case 2:
                  tasksAndStatisticsMenu();
                  break;
            case 3:
                  ListUserAccounts->sort(
                              ServiceUserAccounts::compUserAccountByUserId);
                  ListTasks->sort(ServiceTasks::compTaskByTaskId);
                  writeFile(UserAccount::PATH_USER_ACCOUNTS,
*ListUserAccounts);
                  writeFile(Task::PATH_TASKS, *ListTasks);
                  writeFile(ServiceUserAccounts::PATH_FREE_USERID,
ListFreeUserId);
                  writeFile(ServiceTasks::PATH_FREE_TASKID, ListFreeTaskId);
                  *ListUserAccounts =
readFile(UserAccount::PATH_USER_ACCOUNTS,
```

```
                              *ListUserAccounts);
                  *ListTasks = readFile(Task::PATH_TASKS, *ListTasks);
                  ListFreeUserId =
readFile(ServiceUserAccounts::PATH_FREE_USERID,
                              ListFreeUserId);
                  ListFreeTaskId = readFile(ServiceTasks::PATH_FREE_TASKID,
                              ListFreeTaskId);
                  system("cls");
                  cout << "Изменения сохранены" << endl;
                  system("pause");
                  break;
            case 0:
                  ListUserAccounts->sort(
                              ServiceUserAccounts::compUserAccountByUserId);
                  ListTasks->sort(ServiceTasks::compTaskByTaskId);
                  break;
            default:
                  showExceptionScreen();
            }
            if (variation == 0)
                  break;
      }
}

void AdministratorScreen::accountManagementMenu() {
      unsigned int variation;
      while (true) {
            system("cls");
            showList(*ListUserAccounts);
            cout<<"----------Меню управления учетными записями пользователя---
-------"<<endl;
            cout << "1.\tДобавить учетную запись пользователя\n"
                        << "2.\tРедактировать учетную запись пользователя\n"
                        << "3.\tУдаление учетной записи пользователя\n"
                        << "4.\tУдаление всех учетных записей пользователей\n"
                        << "5.\tСортировка по возрастанию/А-Я\n"
                        << "6.\tСортировка по убыванию/Я-А\n" << "7.\tПоиск\n"
                        << "0.\tВернуться" << endl;
            variation = inputUnsignedInt(cin);
            switch (variation) {
            case 1:
                  ServiceUserAccounts::addUserAccount(*ListUserAccounts,
ListFreeUserId);
                  break;
            case 2:
                  editingUserAccountMenu();
                  break;
            case 3:

      ServiceUserAccounts::deleteUserAccountAndTasks(*ListUserAccounts,
                              *ListTasks, ListFreeUserId, ListFreeTaskId);
                  break;
            case 4:
                  ServiceUserAccounts::deleteAllUserAccountsAndTasks(
                              *ListUserAccounts, *ListTasks, ListFreeUserId,
                              ListFreeTaskId);
                  break;
            case 5:
                  sortInAscendingOrderUserAccountsMenu();
                  break;
            case 6:
                  sortInDecreasingOrderUserAccountsMenu();
                  break;
            case 7:
```

```
                  findUserAccountMenu();
                  break;
            case 0:
                  break;
            default:
                  showExceptionScreen();
            }
            if (variation == 0)
                  break;
      }
}

void AdministratorScreen::tasksAndStatisticsMenu() {
      unsigned int variation;
      while (true) {
            system("cls");
            showList(*ListTasks);
            cout<<"----------Меню управления заданиймми----------"<<endl;
            cout << "1.\tДобавить задание\n" << "2.\tРедактировать задание\n"
                        << "3.\tУдаление задания\n" << "4.\tУдаление всех
заданий\n"
                        << "5.\tСортировка по возрастанию/А-Я\n"
                        << "6.\tСортировка по убыванию/Я-А\n" << "7.\tПоиск\n"
                        << "8.\tСписок просроченных заданий\n"
                        << "9.\tСписок заданий у заданного сотрудника\n"
                        << "10.\tСписок заданий, до истечения срока выполнения
которых осталось X дней\n"
                        << "0.\tВернуться" << endl;
            variation = inputUnsignedInt(cin);
            switch (variation) {
            case 1:
                  ServiceTasks::addTask(*ListTasks, *ListUserAccounts,
ListFreeTaskId);
                  break;
            case 2:
                  editingTaskMenu();
                  break;
            case 3:
                  ServiceTasks::deleteTask(*ListTasks, ListFreeTaskId);
                  break;
            case 4:
                  ServiceTasks::deleteAllTask(*ListTasks, ListFreeTaskId);
                  break;
            case 5:
                  sortInAscendingOrderTasksMenu();
                  break;
            case 6:
                  sortInDecreasingOrderTasksMenu();
                  break;
            case 7:
                  findTaskMenu();
                  break;
            case 8:
                  ServiceTasks::showOverdueTasks(*ListTasks);
                  break;
            case 9:
                  ServiceTasks::showTasksByUserId(*ListTasks);
                  break;
            case 10:
                  ServiceTasks::showOverdueTasksToThreeDays(*ListTasks);
                  break;
            case 0:
                  break;
            default:
```

```
                  showExceptionScreen();
            }
            if (variation == 0)
                  break;
      }
}

void AdministratorScreen::editingUserAccountMenu() {
      system("cls");
      cout
                  << "Введите индентификатор пользователя, данные которого
необходимо изменить"
                  << endl;
      unsigned int inputUserId = inputUnsignedInt(cin);
      ListUserAccounts->sort(ServiceUserAccounts::compUserAccountByUserId);
      list<UserAccount>::iterator findUserAccount =
                  ServiceUserAccounts::findUserAccountByUserId(
                              ListUserAccounts->begin(), ListUserAccounts-
>end(),
                              inputUserId);
      if (findUserAccount != ListUserAccounts->end()) {
            cout
                        << "Подтвердите редактирование этой учетной записи
пользователя (пароль)\n"
                        << (*findUserAccount) << endl;
            string passwordAccount(inputString(cin));
            if (HashPasswordWithSal()(passwordAccount,
                        (*findUserAccount).getSurname())
                        == (*findUserAccount).getPassword()) {
                  unsigned int variation;
                  while (true) {
                        system("cls");
                        cout << (*findUserAccount) << endl;
                        cout<<"----------Меню редактирования учетных записей
пользователей----------"<<endl;
                        cout << "1.\tФамилию Имя Отчество учетной записи
пользователя\n"
                                    << "2.\tОтдел учетной записи
пользователя\n"
                                    << "3.\tПароль учетной записи
пользователя\n"
                                    << "4.\tВсе данные об учетной записи
пользователя\n"
                                    << "0.\tВернуться" << endl;
                        variation = inputUnsignedInt(cin);
                        switch (variation) {
                        case 1:

      ServiceUserAccounts::editingSNMUserAccount(findUserAccount,
                                          passwordAccount);
                              ListUserAccounts->sort();
                              break;
                        case 2:

      ServiceUserAccounts::editingDepartmentUserAccount(
                                          findUserAccount);

                              break;
                        case 3:
                              ServiceUserAccounts::editingPasswordUserAccount(
                                          findUserAccount, passwordAccount);
                              break;
                        case 4:
                              ServiceUserAccounts::editingAllDataUserAccount(
```

```
                                          findUserAccount);
                              ListUserAccounts->sort();
                              break;
                        case 0:
                              break;
                        default:
                              showExceptionScreen();
                        }
                        if (variation == 0)
                              break;
                  }
            } else
                  showExceptionScreen();
      } else
            showNotFindData();
}

void AdministratorScreen::editingTaskMenu() {
      system("cls");
      cout << "Введите индентификатор заданя, данные которого необходимо
изменить"
                  << endl;
      unsigned int inputTaskId = inputUnsignedInt(cin);
      ListTasks->sort(ServiceTasks::compTaskByTaskId);
      list<Task>::iterator findTask = ServiceTasks::findTaskByTaskId(
                  ListTasks->begin(), ListTasks->end(), inputTaskId);
      list<UserAccount>::iterator activeAccount = (*ListUserAccounts).begin();
      if (findTask != ListTasks->end()) {
            cout << "Подтвердите редактирование этого задания (да/нет)\n"
                        << (*findTask) << endl;
            string confirmation(inputString(cin));
            if (confirmation == "да") {
                  unsigned int variation;
                  while (true) {
                        system("cls");
                        cout << (*findTask) << endl;
                        cout<<"----------Меню редактирования заданий----------
"<<endl;
                        cout << "1.\tЗадание\n"
                                    << "2.\tПренадлежность задания к
пользователю\n"
                                    << "3.\tПланируемую дату сдачи\n"
                                    << "4.\tДату сдачи задания\n" <<
"5.\tПримечание\n"
                                    << "6.\tВсе данные о задании\n" <<
"0.\tВернуться"
                                    << endl;
                        variation = inputUnsignedInt(cin);
                        switch (variation) {
                        case 1:
                              ServiceTasks::editingContentTask(findTask);
                              break;
                        case 2:
                              ServiceTasks::editingUserIdTask(findTask,
                                          *ListUserAccounts);
                              break;
                        case 3:
                              ServiceTasks::editingPlannedDateTask(findTask);
                              break;
                        case 4:

      ServiceTasks::editingDateDeliveryTask(activeAccount,
                                          findTask);
                              break;
```

```
                        case 5:
                              ServiceTasks::editingNotationTask(findTask);
                              break;
                        case 6:
                              ServiceTasks::editingAllDataTask(findTask,
                                          *ListUserAccounts);
                              break;
                        case 0:
                              break;
                        default:
                              showExceptionScreen();
                        }
                        if (variation == 0)
                              break;
                  }
            }
            if (confirmation != "да" && confirmation != "нет")
                  showExceptionScreen();
      } else
            showNotFindData();
}

void AdministratorScreen::sortInAscendingOrderUserAccountsMenu() {
      unsigned int variation;
      system("cls");
      cout<<"-----Меню сортировки списка учетных записей пользователей по
возрастанию-----"<<endl;
      cout << "1.\tПо идентификатору\n" << "2.\tПо фамилии\n" << "3.\tПо
имени\n"
                  << "4.\tПо отчеству\n" << "5.\tПо отделам\n" <<
"0.\tВернуться"
                  << endl;
      variation = inputUnsignedInt(cin);
      switch (variation) {
      case 1:
            ListUserAccounts-
>sort(ServiceUserAccounts::compUserAccountByUserId);
            break;
      case 2:
            ListUserAccounts->sort();
            break;
      case 3:
            ListUserAccounts-
>sort(ServiceUserAccounts::compUserAccountByName);
            break;
      case 4:
            ListUserAccounts-
>sort(ServiceUserAccounts::compUserAccountByMidname);
            break;
      case 5:
            ListUserAccounts->sort(
                        ServiceUserAccounts::compUserAccountByDepartment);
            break;
      case 0:
            break;
      default:
            showExceptionScreen();
      }
}

void AdministratorScreen::sortInAscendingOrderTasksMenu() {
      unsigned int variation;
      system("cls");
      cout<<"----------Меню сортировки списка заданий по возрастанию----------
```

```
"<<endl;
	cout << "1.\tПо идентификатору\n" << "2.\tПо заданию\n"
			<< "3.\tПо индексу сотрудника, которому выдано задание\n"
			<< "4.\tПо дате выдачи задания\n"
			<< "5.\tПо дате планируемой сдачи задания\n"
			<< "6.\tПо дате сдачи задания\n" << "7.\tПо примечанию\n"
			<< "0.\tВернуться" << endl;
	variation = inputUnsignedInt(cin);
	switch (variation) {
	case 1:
		ListTasks->sort(ServiceTasks::compTaskByTaskId);
		break;
	case 2:
		ListTasks->sort(ServiceTasks::compTaskByContent);
		break;
	case 3:
		ListTasks->sort();
		break;
	case 4:
		ListTasks->sort(ServiceTasks::compTaskByAssignmentDate);
		break;
	case 5:
		ListTasks->sort(ServiceTasks::compTaskByPlannedDate);
		break;
	case 6:
		ListTasks->sort(ServiceTasks::compTaskByDateDelivery);
		break;
	case 7:
		ListTasks->sort(ServiceTasks::compTaskByNotation);
		break;
	case 0:
		break;
	default:
		showExceptionScreen();
	}
}

void AdministratorScreen::sortInDecreasingOrderUserAccountsMenu() {
	unsigned int variation;
	system("cls");
	cout<<"-----Меню сортировки списка учетных записей пользователей по
убыванию-----"<<endl;
	cout << "1.\tПо идентификатору\n" << "2.\tПо фамилии\n"
			<< "3.\tПо имени\n" << "4.\tПо отчеству\n" << "5.\tПо
отделам\n"
			<< "0.\tВернуться" << endl;
	variation = inputUnsignedInt(cin);
	switch (variation) {
	case 1:
		ListUserAccounts->sort(
				ServiceUserAccounts::recompUserAccountByUserId);
		break;
	case 2:
		ListUserAccounts->sort(
				ServiceUserAccounts::recompUserAccountBySurname);
		break;
	case 3:
		ListUserAccounts->sort(
				ServiceUserAccounts::recompUserAccountByName);
		break;
	case 4:
		ListUserAccounts->sort(
				ServiceUserAccounts::recompUserAccountByMidname);
		break;
```

```
	case 5:
		ListUserAccounts->sort(
				ServiceUserAccounts::recompUserAccountByDepartment);
		break;
	case 0:
		break;
	default:
		showExceptionScreen();
	}
}

void AdministratorScreen::sortInDecreasingOrderTasksMenu(){
	unsigned int variation;
	system("cls");
	cout<<"----------Меню сортировки списка заданий по убыванию----------
"<<endl;
	cout << "1.\tПо идентификатору\n" << "2.\tПо заданию\n"
			<< "3.\tПо по индексу сотрудника, которому выдано задание\n"
			<< "4.\tПо дате выдачи задания\n"
			<< "5.\tПо дате планируемой сдачи задания\n"
			<< "6.\tПо дате сдачи задания\n" << "7.\tПо примечанию\n"
			<< "0.\tВернуться" << endl;
	variation = inputUnsignedInt(cin);
	switch (variation) {
	case 1:
		ListTasks->sort(ServiceTasks::recompTaskByTaskId);
		break;
	case 2:
		ListTasks->sort(ServiceTasks::recompTaskByContent);
		break;
	case 3:
		ListTasks->sort(ServiceTasks::recompTaskByUserId);
		break;
	case 4:
		ListTasks->sort(ServiceTasks::recompTaskByAssignmentDate);
		break;
	case 5:
		ListTasks->sort(ServiceTasks::recompTaskByPlannedDate);
		break;
	case 6:
		ListTasks->sort(ServiceTasks::recompTaskByDateDelivery);
		break;
	case 7:
		ListTasks->sort(ServiceTasks::recompTaskByNotation);
		break;
	case 0:
		break;
	default:
		showExceptionScreen();
	}
}

void AdministratorScreen::findUserAccountMenu() {
	unsigned int variation;
	while (true) {
		system("cls");
		ListUserAccounts-
>sort(ServiceUserAccounts::compUserAccountByUserId);
		cout<<"----------Меню поиска учетных записей пользователей--------
--"<<endl;
		cout << "1.\tПо идентификатору\n" << "2.\tПо фамилии\n"
				<< "3.\tПо имени\n" << "4.\tПо отчеству\n" << "5.\tПо
отделам\n"
				<< "0.\tВернуться" << endl;
```

```
            variation = inputUnsignedInt(cin);
            list<UserAccount> findListUserAccounts;
            list<UserAccount>::iterator begin = ListUserAccounts->begin();
            list<UserAccount>::iterator end = ListUserAccounts->end();
            string str;
            switch (variation) {
            case 1:
                  system("cls");
                  unsigned int userId;
                  cout << "Введите идентификатор для поиска нужной записи" <<
endl;
                  userId = inputUnsignedInt(cin);
                  while (true) {
                        begin =
ServiceUserAccounts::findUserAccountByUserId(begin, end,
                                    userId);
                        if (begin == end)
                              break;
                        findListUserAccounts.push_back(*begin);
                        begin++;
                  }
                  if (findListUserAccounts.size() != 0) {
                        showList(findListUserAccounts);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 2:
                  system("cls");
                  cout
                              << "Введите фамилию (начало/конец/часть фамилии)
для поиска нужной записи"
                              << endl;
                  str = inputString(cin);
                  while (true) {
                        begin =
ServiceUserAccounts::findUserAccountBySurname(begin,
                                    end, str);
                        if (begin == end)
                              break;
                        findListUserAccounts.push_back(*begin);
                        begin++;
                  }
                  if (findListUserAccounts.size() != 0) {
                        showList(findListUserAccounts);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 3:
                  system("cls");
                  cout
                              << "Введите имя (начало/конец/часть имени) для
поиска нужной записи"
                              << endl;
                  str = inputString(cin);
                  while (true) {
                        begin =
ServiceUserAccounts::findUserAccountByName(begin, end,
                                    str);
                        if (begin == end)
                              break;
                        findListUserAccounts.push_back(*begin);
                        begin++;
```

```
                  }
                  if (findListUserAccounts.size() != 0) {
                        showList(findListUserAccounts);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 4:
                  system("cls");
                  cout
                              << "Введите отчество (начало/конец/часть
отчества) для поиска нужной записи"
                              << endl;
                  str = inputString(cin);
                  while (true) {
                        begin =
ServiceUserAccounts::findUserAccountByMidname(begin,
                                    end, str);
                        if (begin == end)
                              break;
                        findListUserAccounts.push_back(*begin);
                        begin++;
                  }
                  if (findListUserAccounts.size() != 0) {
                        showList(findListUserAccounts);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 5:
                  system("cls");
                  cout
                              << "Введите отдела (начало/конец/часть отдела)
для поиска нужной записи"
                              << endl;
                  str = inputString(cin);
                  while (true) {
                        begin =
ServiceUserAccounts::findUserAccountByDepartment(begin,
                                    end, str);
                        if (begin == end)
                              break;
                        findListUserAccounts.push_back(*begin);
                        begin++;
                  }
                  if (findListUserAccounts.size() != 0) {
                        showList(findListUserAccounts);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 0:
                  break;
            default:
                  showExceptionScreen();
            }
            if (variation == 0)
                  break;
            system("pause");
      }
}

void AdministratorScreen::findTaskMenu(){
      unsigned int variation;
```

```
      while (true) {
            system("cls");
            ListTasks->sort(ServiceTasks::compTaskByTaskId);
            cout<<"----------Меню поиска заданий----------"<<endl;
            cout << "1.\tПо идентификатору\n" << "2.\tПо заданию\n"
                        << "3.\tПо по индексу сотрудника, которому выдано
задание\n"
                        << "4.\tПо дате выдачи задания\n"
                        << "5.\tПо дате планируемой сдачи задания\n"
                        << "6.\tПо дате сдачи задания\n" << "7.\tПо
примечанию\n"
                        << "0.\tВернуться" << endl;
            variation = inputUnsignedInt(cin);
            list<Task> findListTasks;
            list<Task>::iterator begin = ListTasks->begin();
            list<Task>::iterator end = ListTasks->end();
            string str;
            Date d;
            switch (variation) {
            case 1:
                  system("cls");
                  unsigned int taskId;
                  cout << "Введите идентификатор для поиска нужной записи" <<
endl;
                  taskId = inputUnsignedInt(cin);
                  while (true) {
                        begin =
ServiceTasks::findTaskByTaskId(begin,end,taskId);
                        if (begin == end)
                              break;
                        findListTasks.push_back(*begin);
                        begin++;
                  }
                  if (findListTasks.size() != 0) {
                        showList(findListTasks);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 2:
                  system("cls");
                  cout
                              << "Введите задание (начало/конец/часть задания)
для поиска нужной записи"
                              << endl;
                  str = inputString(cin);
                  while (true) {
                        begin =
ServiceTasks::findTaskByContent(begin,end,str);
                        if (begin == end)
                              break;
                        findListTasks.push_back(*begin);
                        begin++;
                  }
                  if (findListTasks.size() != 0) {
                        showList(findListTasks);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 3:
                  system("cls");
                  unsigned int userId;
                  cout
```

```
                              << "Введите идентификатор для поиска нужной
записи"
                              << endl;
                  userId = inputUnsignedInt(cin);
                  while (true) {
                        begin =
ServiceTasks::findTaskByUserId(begin,end,userId);
                        if (begin == end)
                              break;
                        findListTasks.push_back(*begin);
                        begin++;
                  }
                  if (findListTasks.size() != 0) {
                        showList(findListTasks);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 4:
                  system("cls");
                  cout
                              << "Введите дату выдачи задания для поиска
нужной записи"
                              << endl;
                  cin >> d;
                  while (true) {
                        begin =
ServiceTasks::findTaskByAssignmentDate(begin,end,d);
                        if (begin == end)
                              break;
                        findListTasks.push_back(*begin);
                        begin++;
                  }
                  if (findListTasks.size() != 0) {
                        showList(findListTasks);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 5:
                  system("cls");
                  cout
                              << "Введите дату планируемой сдачи задания для
поиска нужной записи"
                              << endl;
                  cin >> d;
                  while (true) {
                        begin =
ServiceTasks::findTaskByPlannedDate(begin,end,d);
                        if (begin == end)
                              break;
                        findListTasks.push_back(*begin);
                        begin++;
                  }
                  if (findListTasks.size() != 0) {
                        showList(findListTasks);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 6:
                  system("cls");
                  cout
                              << "Введите дату планируемой сдачи задания для
```

```
поиска нужной записи"
                              << endl;
                  cin >> d;
                  while (true) {
                        begin =
ServiceTasks::findTaskByDateDelivery(begin,end,d);
                        if (begin == end)
                              break;
                        findListTasks.push_back(*begin);
                        begin++;
                  }
                  if (findListTasks.size() != 0) {
                        showList(findListTasks);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 7:
                  system("cls");
                  cout
                              << "Введите примечание (начало/конец/часть
примечания) для поиска нужной записи"
                              << endl;
                  str = inputString(cin);
                  while (true) {
                        begin =
ServiceTasks::findTaskByNotation(begin,end,str);
                        if (begin == end)
                              break;
                        findListTasks.push_back(*begin);
                        begin++;
                  }
                  if (findListTasks.size() != 0) {
                        showList(findListTasks);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 0:
                  break;
            default:
                  showExceptionScreen();
            }
            if (variation == 0)
                  break;
            system("pause");
      }
}
```

Табица 34 – Исходный код в файле UserScreen.h

UserScreen.h

```
#ifndef SCREENS_USERSCREEN_H_
#define SCREENS_USERSCREEN_H_
#include <iostream>
#include <string>
#include <list>
#include "../Data_Classes/Task.h"
#include "../Services/ServiceData.h"
#include "../Services/ServiceTasks.h"
using namespace std;

class UserScreen {
```

```
private:
	list<Task> *ListTasks;
	list<Task> ListTasksBuff;
	list<UserAccount>::iterator *activeAccount;
	static bool created;

	UserScreen(list<Task>&, list<Task>&, list<UserAccount>::iterator&);
	~UserScreen();
	UserScreen(const UserScreen&);
	UserScreen& operator=(const UserScreen&);
	void sortInAscendingOrderTasksMenu();
	void sortInDecreasingOrderTasksMenu();
	void findTaskMenu();
public:
	static UserScreen& instance(list<Task>&, list<UserAccount>::iterator&);
	void showUserScreen();
};

#endif /* SCREENS_USERSCREEN_H_ */
```

Табица 35 – Исходный код в файле UserScreen.cpp

UserScreen.cpp

```
#include "UserScreen.h"

bool UserScreen::created = false;

UserScreen::UserScreen(list<Task> &ListTasks, list<Task> &ListTasksBuff,
list<UserAccount>::iterator &activeAccount) {
	this->ListTasks = &ListTasks;
	this->ListTasksBuff = ListTasksBuff;
	this->activeAccount = &activeAccount;
	this->created = true;
}

UserScreen::~UserScreen() {
}

UserScreen& UserScreen::instance(list<Task> &ListTasks,
		list<UserAccount>::iterator &activeAccount) {
	list<Task> ListTasksBuff;
	list<Task>::iterator begin = ListTasks.begin();
	list<Task>::iterator end = ListTasks.end();
	while (true) {
		begin = ServiceTasks::findTaskByUserId(begin, end,
(*activeAccount).getUserId());
		if (begin == end)
			break;
		ListTasksBuff.push_back(*begin);
		begin++;
	}
	static UserScreen *inst;
	if (!created)
		inst = new UserScreen(ListTasks, ListTasksBuff, activeAccount);
	return *inst;
}

void UserScreen::showUserScreen() {
	unsigned int variation;
	while (true) {
		system("cls");
		showList(ListTasksBuff);
		cout<<"----------Главное меню пользователя----------"<<endl;
```

```
            cout<<"1.\tУведомить о выполнении задания\n"
                        <<"2.\tСортировка по возрастанию/А-Я\n"
                        <<"3.\tСортировка по убыванию/Я-А\n"
                        <<"4.\tПоиск\n"
                        <<"5.\tСписок просроченых задания\n"
                        <<"6.\tСписок заданий, до истечения срока выполнения
которых осталось 3 дней\n"
                        <<"0.\tВыйти и сохранить изменения"
                        <<endl;
            variation = inputUnsignedInt(cin);
            switch(variation){
            case 1:
            {
                  system("cls");
                  cout << "Введите индентификатор заданя, данные которого
необходимо изменить"
                              << endl;
                  unsigned int inputTaskId = inputUnsignedInt(cin);
                  ListTasksBuff.sort(ServiceTasks::compTaskByTaskId);
                  list<Task>::iterator findTask =
ServiceTasks::findTaskByTaskId(
                              ListTasksBuff.begin(), ListTasksBuff.end(),
inputTaskId);
                  if (findTask != ListTasksBuff.end()) {
                        cout << "Подтвердите редактирование этого задания
(да/нет)\n"
                                    << (*findTask) << endl;
                        string confirmation(inputString(cin));
                        if (confirmation == "да") {

      ServiceTasks::editingDateDeliveryTask(*activeAccount,findTask);
                        }
                        if (confirmation != "да" && confirmation != "нет")
                              showExceptionScreen();
                  }
                  else
                        showNotFindData();
            }
                  break;
            case 2:
                  sortInAscendingOrderTasksMenu();
                  break;
            case 3:
                  sortInDecreasingOrderTasksMenu();
                  break;
            case 4:
                  findTaskMenu();
                  break;
            case 5:
                  ServiceTasks::showOverdueTasks(ListTasksBuff);
                  break;
            case 6:
                  ServiceTasks::showOverdueTasksToThreeDays(ListTasksBuff);
                  break;
            case 0:
            {
                  list<Task>::iterator findEditTask;
                  for(Task t:ListTasksBuff){
                        findEditTask =
ServiceTasks::findTaskByTaskId((*ListTasks).begin(),(*ListTasks).end(),t.getT
askId());
                        *findEditTask = t;
                  }
                  ListTasks->sort(ServiceTasks::compTaskByTaskId);
```

```
                  writeFile(Task::PATH_TASKS, *ListTasks);
                  break;
            }
            default:
                  showExceptionScreen();
            }
            if (variation == 0)
                  break;
      }
}

void UserScreen::sortInAscendingOrderTasksMenu(){
      unsigned int variation;
      system("cls");
      cout<<"----------Меню для сортировки заданий в порядке возростания------
----"<<endl;
      cout << "1.\tПо идентификатору\n" << "2.\tПо заданию\n"
                  << "3.\tПо по индексу сотрудника, которому выдано задание\n"
                  << "4.\tПо дате выдачи задания\n"
                  << "5.\tПо дате планируемой сдачи задания\n"
                  << "6.\tПо дате сдачи задания\n" << "7.\tПо примечанию\n"
                  << "0.\tВернуться" << endl;
      variation = inputUnsignedInt(cin);
      switch (variation) {
      case 1:
            ListTasksBuff.sort(ServiceTasks::compTaskByTaskId);
            break;
      case 2:
            ListTasksBuff.sort(ServiceTasks::compTaskByContent);
            break;
      case 3:
            ListTasksBuff.sort();
            break;
      case 4:
            ListTasksBuff.sort(ServiceTasks::compTaskByAssignmentDate);
            break;
      case 5:
            ListTasksBuff.sort(ServiceTasks::compTaskByPlannedDate);
            break;
      case 6:
            ListTasksBuff.sort(ServiceTasks::compTaskByDateDelivery);
            break;
      case 7:
            ListTasksBuff.sort(ServiceTasks::compTaskByNotation);
            break;
      case 0:
            break;
      default:
            showExceptionScreen();
      }
}

void UserScreen::sortInDecreasingOrderTasksMenu(){
      unsigned int variation;
      system("cls");
      cout<<"----------Меню для сортировки заданий в порядке убывания---------
-"<<endl;
      cout << "1.\tПо идентификатору\n" << "2.\tПо заданию\n"
                  << "3.\tПо по индексу сотрудника, которому выдано задание\n"
                  << "4.\tПо дате выдачи задания\n"
                  << "5.\tПо дате планируемой сдачи задания\n"
                  << "6.\tПо дате сдачи задания\n" << "7.\tПо примечанию\n"
                  << "0.\tВернуться" << endl;
      variation = inputUnsignedInt(cin);
```

```
	switch (variation) {
	case 1:
		ListTasksBuff.sort(ServiceTasks::recompTaskByTaskId);
		break;
	case 2:
		ListTasksBuff.sort(ServiceTasks::recompTaskByContent);
		break;
	case 3:
		ListTasksBuff.sort(ServiceTasks::recompTaskByUserId);
		break;
	case 4:
		ListTasksBuff.sort(ServiceTasks::recompTaskByAssignmentDate);
		break;
	case 5:
		ListTasksBuff.sort(ServiceTasks::recompTaskByPlannedDate);
		break;
	case 6:
		ListTasksBuff.sort(ServiceTasks::recompTaskByDateDelivery);
		break;
	case 7:
		ListTasksBuff.sort(ServiceTasks::recompTaskByNotation);
		break;
	case 0:
		break;
	default:
		showExceptionScreen();
	}
}

void UserScreen::findTaskMenu(){

	unsigned int variation;
	while (true) {
		system("cls");
		cout<<"----------Меню для поиска заданий----------"<<endl;
		cout << "1.\tПо идентификатору\n" << "2.\tПо заданию\n"
				<< "3.\tПо по индексу сотрудника, которому выдано
задание\n"
				<< "4.\tПо дате выдачи задания\n"
				<< "5.\tПо дате планируемой сдачи задания\n"
				<< "6.\tПо дате сдачи задания\n" << "7.\tПо
примечанию\n"
				<< "0.\tВернуться" << endl;
		variation = inputUnsignedInt(cin);
		list<Task> findListTasks;
		list<Task>::iterator begin = ListTasksBuff.begin();
		list<Task>::iterator end = ListTasksBuff.end();
		string str;
		Date d;
		switch (variation) {
		case 1:
			system("cls");
			unsigned int taskId;
			cout << "Введите идентификатор для поиска нужной записи" <<
endl;
			taskId = inputUnsignedInt(cin);
			while (true) {
				begin =
ServiceTasks::findTaskByTaskId(begin,end,taskId);
				if (begin == end)
					break;
				findListTasks.push_back(*begin);
				begin++;
			}
```

```
                  if (findListTasks.size() != 0) {
                        showList(findListTasks);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 2:
                  system("cls");
                  cout
                              << "Введите задание (начало/конец/часть задания)
для поиска нужной записи"
                              << endl;
                  str = inputString(cin);
                  while (true) {
                        begin =
ServiceTasks::findTaskByContent(begin,end,str);
                        if (begin == end)
                              break;
                        findListTasks.push_back(*begin);
                        begin++;
                  }
                  if (findListTasks.size() != 0) {
                        showList(findListTasks);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 3:
                  system("cls");
                  unsigned int userId;
                  cout
                              << "Введите идентификатор для поиска нужной
записи"
                              << endl;
                  userId = inputUnsignedInt(cin);
                  while (true) {
                        begin =
ServiceTasks::findTaskByUserId(begin,end,userId);
                        if (begin == end)
                              break;
                        findListTasks.push_back(*begin);
                        begin++;
                  }
                  if (findListTasks.size() != 0) {
                        showList(findListTasks);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 4:
                  system("cls");
                  cout
                              << "Введите дату выдачи задания для поиска
нужной записи"
                              << endl;
                  cin >> d;
                  while (true) {
                        begin =
ServiceTasks::findTaskByAssignmentDate(begin,end,d);
                        if (begin == end)
                              break;
                        findListTasks.push_back(*begin);
                        begin++;
                  }
```

```
                  if (findListTasks.size() != 0) {
                        showList(findListTasks);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 5:
                  system("cls");
                  cout
                              << "Введите дату планируемой сдачи задания для
поиска нужной записи"
                              << endl;
                  cin >> d;
                  while (true) {
                        begin =
ServiceTasks::findTaskByPlannedDate(begin,end,d);
                        if (begin == end)
                              break;
                        findListTasks.push_back(*begin);
                        begin++;
                  }
                  if (findListTasks.size() != 0) {
                        showList(findListTasks);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 6:
                  system("cls");
                  cout
                              << "Введите дату планируемой сдачи задания для
поиска нужной записи"
                              << endl;
                  cin >> d;
                  while (true) {
                        begin =
ServiceTasks::findTaskByDateDelivery(begin,end,d);
                        if (begin == end)
                              break;
                        findListTasks.push_back(*begin);
                        begin++;
                  }
                  if (findListTasks.size() != 0) {
                        showList(findListTasks);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 7:
                  system("cls");
                  cout
                              << "Введите примечание (начало/конец/часть
примечания) для поиска нужной записи"
                              << endl;
                  str = inputString(cin);
                  while (true) {
                        begin =
ServiceTasks::findTaskByNotation(begin,end,str);
                        if (begin == end)
                              break;
                        findListTasks.push_back(*begin);
                        begin++;
                  }
                  if (findListTasks.size() != 0) {
```

```
                        showList(findListTasks);
                        system("pause");
                  } else
                        showNotFindData();
                  break;
            case 0:
                  break;
            default:
                  showExceptionScreen();
            }
            if (variation == 0)
                  break;
            system("pause");
      }
}
```

Табица 36 – Исходный код в файле ControllerIOFStream.h

ControllerIOFStream.h

```
#ifndef COMTROLLERS_CONTROLLERIOFSTREAM_H_
#define COMTROLLERS_CONTROLLERIOFSTREAM_H_

#include <string>
#include <list>
#include <fstream>
using namespace std;

template<class T>
list<T> readFile(string path, list<T>& List) {
      List.clear();
      ifstream inputStream(path);
      size_t size;
      T buf;
      inputStream >> size;
      for (size_t i = 0; i < size; i++) {
            inputStream >> buf;
            List.push_back(buf);
      }
      inputStream.close();
      return List;
}

template<class T>
void writeFile(string path, list<T>& List) {
      ofstream outputStream(path);
      outputStream << List.size() << "\n";
      for (T ptr : List) {
            outputStream << ptr << endl;
      }
      outputStream.close();
}

#endif /* COMTROLLERS_CONTROLLERIOFSTREAM_H_ */
```

Табица 37 – Исходный код в файле ControllerIStream.h

ControllerIStream.h

```
#ifndef COMTROLLERS_CONTROLLERISTREAM_H_
#define COMTROLLERS_CONTROLLERISTREAM_H_

#include <iostream>
#include <string>
#include <list>
```

```
#include <cstdlib>
#include <limits>
using namespace std;

string inputString(istream&);
string inputLine(istream&);
unsigned int inputUnsignedInt(istream&);
short int inputShortInt(istream&);

#endif /* COMTROLLERS_CONTROLLERISTREAM_H_ */
```

Табица 38 – Исходный код в файле ControllerIStream.cpp

ControllerIStream.cpp

```
#include "ControllerIStream.h"

string inputString(istream& stream){
	string str;
	cout<<">> ";
	stream>>str;
	stream.clear();
	cin.ignore(numeric_limits<streamsize>::max(), '\n');
	return str;
}

string inputLine(istream& stream){
	string str;
	cout<<">> ";
	getline(stream,str);
	return str;
}

unsigned int inputUnsignedInt(istream& stream){
	return atoi(inputString(stream).c_str());
}

short int inputShortInt(istream& stream){
	return atoi(inputString(stream).c_str());
}
```

Табица 39 – Исходный код в файле HashPasswordWithSal.h

HashPasswordWithSal.h

```
#ifndef HASHPASSWORDWITHSAL_H_
#define HASHPASSWORDWITHSAL_H_

#include <string>
#include <functional>
using namespace std;

class HashPasswordWithSal{
public:
	string operator()(const string password, const string sal) const{
		size_t h1 = hash<string>()(password);
		size_t h2 = hash<string>()(sal);
		return to_string(h1^(h2 << 1));
	}
};

#endif /* HASHPASSWORDWITHSAL_H_ */
```

Табица 40 – Данные в файле FreeUserId.txt на начале работы

FreeUserId.txt
0

Табица 41 – Данные в файле FreeTaskId.txt на начале работы

FreeTaskId.txt
0

Табица 42 – Данные в файле UserAccounts.txt на начале работы

UserAccounts.txt
1 0 3024866781 Бабанин Дмитрий Сергеевич Начальник

Табица 43 – Данные в файле Tasks.txt на начале работы

Tasks.txt
0

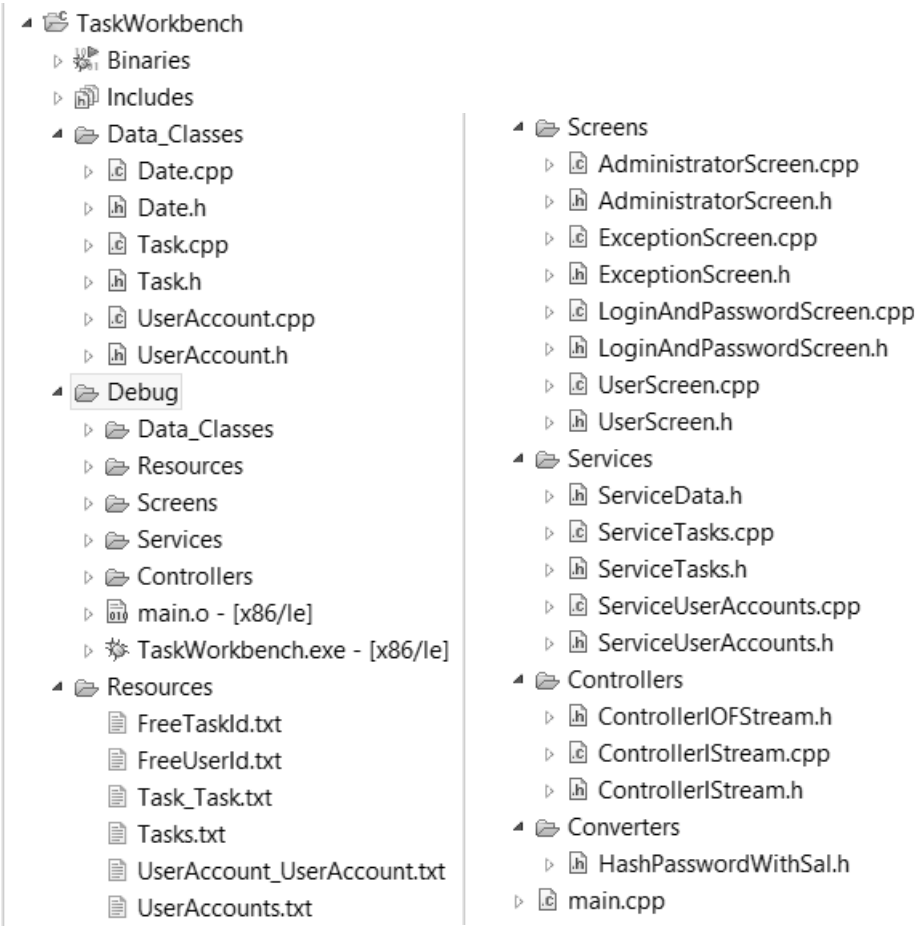

Рисунок В.1 – Дерево расположения файлов проекта в IDE Eclipse Cpp Oxygen