ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ИНСТИТУТ
МЕЖДУНАРОДНЫХ ОТНОШЕНИЙ (УНИВЕРСИТЕТ)
МИНИСТЕРСТВО ИНОСТРАННЫХ ДЕЛ РОССИЙСКОЙ ФЕДЕРАЦИИ»
ОДИНЦОВСКИЙ ФИЛИАЛ

ФАКУЛЬТЕТ ФИНАНСОВОЙ ЭКОНОМИКИ

КАФЕДРА МАТЕМАТИЧЕСКИХ МЕТОДОВ
И БИЗНЕС-ИНФОРМАТИКИ

ДИСЦИПЛИНА: АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ И ПРОГРАММИРОВАНИЕ

КУРСОВАЯ РАБОТА

на тему:

“ Программная реализация работы отдела кадров предприятия на основе STLязыка C++”

НАПРАВЛЕНИЕ ПОДГОТОВКИ

38.03.05 «Бизнес-информатика»

ПРОФИЛЬ ПОДГОТОВКИ

«Информационные технологии в международном бизнесе»

Выполнила: Гордеева Мария Игоревна

студентка 1 курса, группы ИТБ-21

очной формы обучения

Научный руководитель:

Ерохин Виктор Викторович

Одинцово 2025

СОДЕРЖАНИЕ

Введение

Целью курсовой работы является разработка простой и практичной программы для автоматизации повседневной деятельности отдела кадров на предприятии. В основу проекта положено создание инструмента для оперативного просмотра, редактирования и систематизации информации о сотрудниках.

Актуальность темы определяется недостатком доступных и удобных решений для ведения кадровой отчётности в современных организациях. Отсутствие таких инструментов приводит к увеличению времени, затрачиваемого на ручную обработку информации, оформление документов и контроль кадровых перемещений. Использование пециализированного программного обеспечения позволяет упростить указанные процессы, повысить точность учёта и сократить затраты времени и ресурсов.

Работа направлена на решение практической задачи — отсутствие универсальных программных средств для эффективной автоматизации учёта персонала в небольших организациях. Разрабатываемая система предназначена для выполнения рутинных задач по ведению справочной информации о сотрудниках и обеспечения удобного доступа к ключевым кадровым данным.

Область исследования охватывает структуру предприятия и особенности документооборота в сфере кадровой деятельности. Особое внимание уделено анализу работы отдела кадров, его основным функциям, типовым процессам и требованиям к системам учёта. На основе проведённого анализа формируется практическое программное решение.

Разрабатываемая программа хранит и обрабатывает следующие данные:

- фамилия, имя, отчество сотрудника;
- возраст;
- должность;
- подразделение (отдел);
- заработная плата (фиксированная, привязанная к должности);
- история кадровых перемещений (переводы по отделам);
- сведения об увольнениях.

Дополнительно реализована функция вывода информации на печать для получения бумажных копий кадровых данных при необходимости.

Цели и задачи

Реализация направлена на создание простой информационной системы, обеспечивающей оперативное получение сведений о персонале и выполнение базовых операций с кадровыми данными.

Для достижения поставленной цели решаются следующие задачи:

1. Изучение принципов работы отдела кадров и выполняемых им функций.
2. Сбор справочной информации, необходимой для проектирования системы.
3. Определение подходящего языка программирования и инструментов реализации.
4. Разработка прикладной программы в соответствии с установленными требованиями.
5. Проведение тестирования разработанного решения.
6. Выполнение отладки, устранение возможных ошибок и доработка функционала при необходимости.

Результатом работы будет программный продукт, предназначенный для автоматизации ключевых процессов учёта персонала и снижения нагрузки на сотрудников кадровой службы.

Средства разработки

Для реализации приложения использован язык программирования C++ благодаря эффективности работы с памятью, широкой поддержке стандартной библиотеки и пригодности для создания консольных приложений. В проекте активно применялась стандартная библиотека шаблонов STL, включая контейнеры map, vector и set.

- map<int, Employee> использовался для хранения базы сотрудников, где каждому табельному номеру соответствует полный набор данных о сотруднике.
- Vector применялся для фиксации списка прогулов — дат отсутствия сотрудника на рабочем месте.
- Set обеспечивал хранение уникального перечня отделов предприятия.

В коде использована библиотека <chrono> для работы с текущей датой, что позволяет фиксировать пропуски сотрудников по актуальному дню. Для корректного отображения русского текста в консоли Windows применялась библиотека <Windows.h>, а настройка кодировки осуществлялась с помощью SetConsoleOutputCP(1251).

Разработка выполнялась в среде Visual Studio, широко используемой для написания программ на C++. Основной задачей программы является создание небольшой информационной системы, предназначенной для автоматизации основных функций отдела кадров. Решение особенно актуально для предприятий с ограниченными ресурсами, где отсутствует возможность внедрения полноценных корпоративных CRM или кадровых систем.

Взаимодействие с программой осуществляется через текстовое меню, отображающееся при запуске. Действия выполняются пошагово: сначала выбирается нужный пункт (например, «нанять сотрудника»), затем вводятся данные (ФИО, должность, отдел и т. д.). Ввод производится с клавиатуры, результаты отображаются на экране. После каждого действия база данных автоматически сохраняется в файл, что предотвращает потерю информации при закрытии программы.

Разработанное приложение объединяет простоту интерфейса, последовательную логику работы и базовую функциональность, достаточную для автоматизации кадрового учёта.

Структурное описание разработки

Программа, которая разработана в рамках данного описания, представляет собой консольное приложение. Программы состоит из 5 файлов, где каждый файл используется для логичного разделения программы.

Основная цель этого консольного предложения заключается в замене ручного труда отдела кадров простой системы учёта сотрудников, которая будет эффективна, понятна и сохранить все данные без лишних усилий.

Разработка выполнена на языке C++ с применением стандартной библиотеки шаблонов (STL) и модульного подхода. Для каждого модуля предусмотрена отдельная логическая функция, что обеспечивает гибкость, удобство сопровождения и потенциал для расширения программы.

Структурное описание консольного приложения включает в себя следующие аспекты:

1. Разделение кода на модули по функциональному назначению (главный модуль, логика работы с кадрами, хранение данных).

2. Использование структуры Employee для хранения информации о каждом сотруднике.

3. Применение стандартных контейнеров STL (map, vector, set) для организации и обработки данных.

4. Реализация класса EmpDict, отвечающего за основные операции с персоналом.

5. Реализация класса SaveLoadData для обеспечения сохранности информации между сессиями работы.

6. Взаимодействие модулей между собой через включение исходных файлов и передачу данных.

7. Поддержка пользовательского интерфейса в виде текстового меню для выбора действий.

Далее будут подробно рассмотрены формы взаимодействия с консольным приложением:

Описание охватывает структуру меню, представленную при запуске программы, доступные варианты действий, а также последовательность выполнения операций. Вся работа пользователя организована через текстовое меню, формируемое в функции main() в файле kadry.cpp.

```
cout << "Добро пожаловать в программу" << endl;
cout << "Выберите действие:" << endl;
cout << "1.Посмотреть список сотрудников" << endl;
cout << "2.Нанять сотрудника" << endl;
cout << "3.Уволить сотрудника" << endl;
cout << "4.Перевести в другой отдел" << endl;
cout << "5.Выйти их программы" << endl;
```

После отображения меню программа ожидает ввода числа, соответствующего выбранному действию. Ввод осуществляется с помощью команды wcin >> action;, после чего выбранный пункт обрабатывается через оператор switch.

```
switch (action)
{
case 1:
    cin.ignore();
    eDict.displayEmp();
    cin.get();
    continue;
    break;
case 2:

    cin.ignore();
    eDict.addEmployee();
    break;
case 3:
    eDict.dismissalEmployee();
    break;
case 4:
    eDict.moveToDepartment();
    break;
case 5:
    return 0;
    break;
default:
    cout << "Введён не корректный идентификатор пункта меню" << endl;
    break;
}
```

Ввод осуществляется последовательно: сначала выбирается действие, затем, если это необходимо, программа предлагает ввести дополнительные

данные (например, имя, отдел, табельный номер и т. д.). На этом этапе используются стандартные операторы cin и getline.

Например, при добавлении сотрудника вызывается метод addEmployee(), в котором пользователь последовательно вводит все данные, нужные для создания записи:

```
cout << "Введите данные сотрудника: " << endl;
cout << "Фамилия: ";
getline(cin, emp.firstName);
cout << "Имя: ";
getline(cin, emp.lastName);
cout << "Отчество: ";
getline(cin, emp.patronimic);
cout << "Возраст: ";
cin >> emp.age;
```

Также предусмотрена проверка уникальности табельного номера и выбор должности и отдела из предложенного списка. Эти действия реализованы через методы enterServiceNumber(), selectPost() и selectDepart().

После выполнения действия, например, после приёма на работу или увольнения, данные сохраняются в файл автоматически через метод saveData() класса SaveLoadData.

Формы взаимодействия организованы по понятной пошаговой логике: отображение меню, ввод команды, выполнение действия, подтверждение или возврат к меню. Такой подход упрощает освоение программы и обеспечивает возможность её использования без специальной технической подготовки.

Функциональное описание разработки

В этой главе подробно рассматривается, какие именно задачи решает созданная программа и как она работает на практике. Основное внимание уделяется тому, как пользователь взаимодействует с системой, какие действия он может выполнять, и как программа на эти действия реагирует.

Программа работает в текстовом интерфейсе: при запуске пользователю предлагается простое меню с выбором действия. Это делает её понятной и доступной для любого человека, даже без большого опыта работы с компьютером. Пользователь может нанимать новых сотрудников, увольнять, переводить их в другие отделы, просматривать текущий состав предприятия, отмечать прогулы и даже отправлять список сотрудников на печать.

Особое внимание в программе уделено сохранению данных. Все изменения — например, добавление нового сотрудника или увольнение — автоматически сохраняются в файл, благодаря чему информация не теряется при закрытии программы.

Функциональное описание файла kadry.cpp:

Файл kadry.cpp содержит основную управляющую часть программы, которая предназначена для автоматизации работы отдела кадров предприятия. Это так называемый файл с функцией main(), с которого начинается выполнение всей программы.

Программа реализует простое консольное меню для управления базой сотрудников. С её помощью можно:

1. просматривать список сотрудников,
2. нанимать новых работников,
3. увольнять сотрудников,
4. переводить их в другие отделы.

Используемые заголовочные файлы:

```
#include <iostream>
#include <string>
#include <locale>
#include <Windows.h>
#include "empdict.h"
```

#include <iostream> — позволяют использовать ввод/вывод и строки.

#include <locale> и #include <Windows.h> — используются для настройки корректного отображения русского языка в консоли Windows.

"empdict.h" — это пользовательский заголовочный файл, в котором описан класс EmpDict (словарь сотрудников), реализующий все основные функции работы с персоналом.

Работа с датой:

```
using namespace std::chrono;

int main()
{
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);

    EmpDict eDict;
    while (true)
    {
        int action{};
        system("cls");
        auto today = year_month_day{ floor<days>(system_clock::now()) };
        std::cout << "Сегодня " << std::format("{:%d.%m.%Y}\n", sys_days{today}) << endl;
```

Программа автоматически определяет текущую дату и отображает её при запуске. Это помогает пользователю видеть, когда он работает с базой данных.

Локализация:

```
{
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);
```

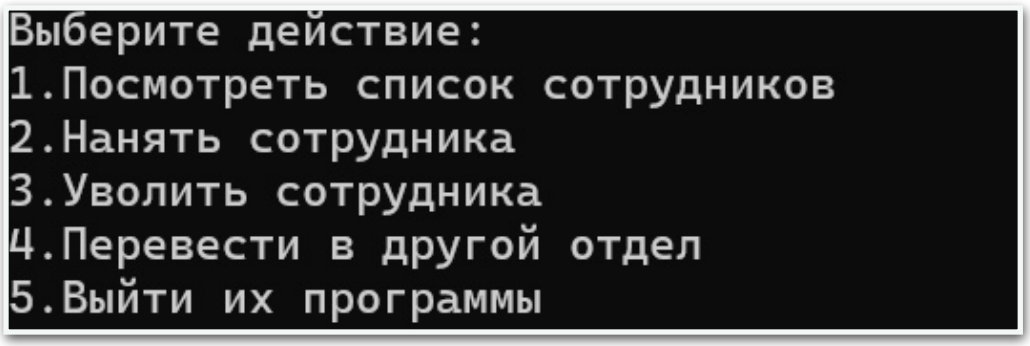
Эти строки нужны, чтобы корректно отображать и вводить русский текст в консоли Windows (кириллица может отображаться с ошибками без этих команд).

Основная логика работы:

Программа построена как бесконечный цикл, который предлагает пользователю меню и ждёт выбора действия. Это цикл while (true) — он будет выполняться до тех пор, пока пользователь явно не выберет выход (пункт 5).

Главное меню:

На экран выводится 5 пунктов:



Далее пользователь выбирает действие, вводя цифру от 1 до 5. Программа анализирует этот ввод с помощью конструкции switch.

Действия в меню:

1. Посмотреть список сотрудников. Вызывает метод eDict.displayEmp(), который выводит всех сотрудников, зарегистрированных в базе.

2. Нанять сотрудника. Вызывает метод eDict.addEmployee(). Программа предложит ввести информацию о новом работнике: ФИО, должность, подразделение и т.д., и добавит его в базу.

3. Уволить сотрудника. Вызывает метод eDict.dismissalEmployee(). Удаляет сотрудника из базы по указанным данным (обычно по фамилии или ID).

4. Перевести в другой отдел. Метод eDict.moveToDepartment(). Меняет информацию о подразделении у выбранного сотрудника.

5. Выйти из программы. Команда return 0; завершает выполнение программы.

Ошибка ввода:

Если пользователь введёт неправильную цифру, программа выдаст сообщение: Введен не корректный идентификатор пункта меню.

```
default:
    cout << "Введён не корректный идентификатор пункта меню" << endl;
    break;
}
```

Использование класса EmpDict

Класс EmpDict отвечает за всю внутреннюю работу с сотрудниками. Он реализован в других файлах (empdict.cpp, empdict.h), а здесь создаётся его объект. Через него вызываются все функции по управлению персоналом.

```
EmpDict eDict;
while (true)
{
```

Функциональное описание файла saveloaddata.h:

Файл saveloaddata.h представляет собой заголовочный файл, в котором описаны структура данных о сотрудниках и интерфейс класса SaveLoadData — отвечающего за сохранение и загрузку информации о персонале в/из файла.

Этот файл играет важную роль в обеспечении долговременного хранения данных. Файл выполняет следующие функции:

- сохранить текущую базу сотрудников в файл;

- загрузить сохранённую базу при следующем запуске программы.

Структура Employee:

В этом файле определяется структура Employee — это основной тип данных, в котором хранится информация об одном сотруднике.

```
struct Employee{
    int number{};

    string firstName{};
    string lastName{};
    string patronimic{};

    int     age{};
    string  post{};
    string  department{};

    bool isDismis = false;
    vector<year_month_day> vTruancy{};
};
```

Пояснение:

int number — уникальный номер (ID) сотрудника;

string firstName, lastName, patronimic — имя, фамилия и отчество;

int age — возраст сотрудника;

string post — должность;

string department — отдел/подразделение, в котором работает сотрудник;

bool isDismis — флаг, уволен ли сотрудник (по умолчанию — false);

vector<year_month_day> vTruancy — список прогулов (даты, когда сотрудник не выходил на работу).

Таким образом, структура Employee содержит всю ключевую информацию о человеке, необходимую отделу кадров.

Класс SaveLoadData:

Класс SaveLoadData реализует механизмы сохранения и загрузки данных из файла, т.е. он обеспечивает функциональность долговременного хранения базы сотрудников между сессиями работы с программой.

Методы класса:

1. SaveLoadData(). Конструктор класса. Пока он не содержит кода, но может использоваться в будущем для инициализации или логирования.

2. bool saveData(map<int, Employee> empDict). Метод сохранения данных. Принимает словарь (map<int, Employee>) — то есть всю текущую базу сотрудников — и сохраняет её в файл. Возвращает true, если сохранение прошло успешно, или false при ошибке.

3. map<int, Employee> loadData(). Метод загрузки данных.
Читает данные из файла и восстанавливает базу сотрудников в виде map<int, Employee>. Это позволяет при следующем запуске программы продолжить работу с уже сохранённой базой.

Назначение map<int, Employee>:

Программа использует ассоциативный контейнер map<int, Employee> из стандартной библиотеки C++ (STL).
Это в некотором роде "словарь", где:

- ключ (int) — уникальный номер сотрудника;

- значение (Employee) — структура с полной информацией об этом человеке.

Использование такого контейнера делает поиск, добавление и удаление сотрудников быстрым и удобным.

Назначение файла в структуре программы:

Файл saveloaddata.h работает совместно с saveloaddata.cpp, в котором реализованы тела этих методов. Вместе они позволяют:

- не потерять данные при закрытии программы;
- быстро восстановить всю базу сотрудников при следующем запуске;
- обеспечить удобную и надёжную работу с данными, важными для отдела кадров.

Функциональное описание файла saveloaddata.cpp:

Файл saveloaddata.cpp реализует методы класса SaveLoadData, объявленного в saveloaddata.h. Этот файл отвечает за постоянное хранение данных о сотрудниках предприятия, то есть за сохранение и загрузку базы данных из файла. Это важная часть программы, она позволяет не потерять информацию после закрытия приложения.

Задачи файла:

Он реализует два главных метода:

1. saveData(...) — сохраняет текущую базу сотрудников в файл;
2. loadData() — загружает ранее сохранённую базу из файла.

Вся информация записывается и читается в бинарном формате, что делает операции быстрее и экономит место.

Описание методов:

1. SaveLoadData::saveData(map<int, Employee> empDict).

Цель этого метода сохранить все данные о сотрудниках в файл database.db.

Поэтапный разбор, как работает метод:

1. Открывается файл для записи в бинарном режиме:

```
ofstream file("database.db", ios::binary);
if(!file)
{
```

2. Проверяется, удалось ли открыть файл. Если нет — выводится ошибка.

3. В файл записывается:

- размер базы (empDict.size()), чтобы при загрузке знать, сколько сотрудников обрабатывать;
- далее по каждому сотруднику вводится индивидуальная информация: его уникальный номер (number)
- ФИО, должность, подразделение возраст;
- флаг увольнения (isDismis);
- список прогулов (vTruancy — даты отсутствия на работе).

Обработка строк:

Для строк сначала записывается длина строки, а затем сама строка. Так при чтении удается точно узнать сколько байт нужно считать.

Обработка дат:

Прогулы (vTruancy) сохраняются как список дат. Каждая дата разбивается на день, месяц и год, которые записываются по отдельности как целые числа (по 4 байта).

В результате, файл database.db содержит всю информацию о сотрудниках и готов для последующей загрузки кадров.

2. SaveLoadData::loadData()

Цель этого метода загрузить данные из файла database.db и восстановить словарь сотрудников.

Поэтапный разбор, как работает метод:

1. Открывается файл на чтение в бинарном режиме:

```
ifstream file("database.db", ios::binary);
if(!file)
```

2. Если файл не найден, программа сообщает об этом и возвращает пустую базу (map<int, Employee>).

3. Из файла читается:

- общее количество сотрудников (map_size);

- по каждому сотруднику:его ключ в словаре (int); все поля структуры Employee: имя, фамилия, отчество; должность, отдел, возраст, флаг увольнения; список прогулов (читаются по три числа: день, месяц, год и собираются в year_month_day).

4. Каждая запись вставляется в map<int, Employee> - это база данных сотрудников в памяти.

В результате, программа восстанавливает всю базу данных из файла и может продолжать с ней работать как с обычной коллекцией map.

Назначение в структуре программы:

Модуль (в паре с заголовочным файлом) используется как часть системы долговременного хранения данных. Логика работы с сотрудниками отделена от логики хранения, что обеспечивает гибкость программы.

Функциональное описание файла empdict.h:

В заголовочном файле empdict.h приведены объявления основного логического класса (EmpDict), в котором реализована бизнес-логика управления базой сотрудников: добавление, удаление, перевод, отображение. Также предусмотрена работа с сохранением и загрузкой данных с использованием объекта SaveLoadData.

Назначение класса EmpDict:

EmpDict — это центральный управляющий класс для работы с кадрами. Имя файла расшифровывается как Employee Dictionary — "словарь сотрудников". Осуществляется управление всей логикой, связанной с обработкой данных о сотрудниках:

- хранению сотрудников;

- вводу данных с консоли;

- отображению данных пользователю;
- работе с файлами.

Состав класса:

Открытые методы (public:)

1. EmpDict() — конструктор.

При создании объекта инициализирует внутренние структуры и загружает сохранённые данные.

2. ~EmpDict() — деструктор.

При завершении работы программы автоматически вызывает сохранение текущих данных в файл.

3. void addEmployee()

Добавляет нового сотрудника. Запрашивает у пользователя данные (ФИО, должность, отдел и т.д.) и записывает их в базу.

4. void dismissalEmployee()

Помечает сотрудника как уволенного. Изменяет флаг isDismis в структуре Employee.

5. void moveToDepartment()

Переводит сотрудника в другой отдел. Обновляет поле department.

6. void displayEmp()

Показывает список сотрудников на экране. Скорее всего, выводит их в виде таблицы с основными данными.

Закрытые поля и методы (private:)

Внутренние данные:

```
map<int, Employee> loadMap{};
ifstream file("database.db", ios::binary);
if(!file)
{
```

map<int, Employee> empDict{};

- Основная база сотрудников.

- Ключ — уникальный номер (ID), значение — структура Employee.

```
map<string, int> post = {{"Генеральный директор", 500000},{"Главный бухгалтер", 400000},{"Бухгалтер", 200000},{"Менеджер", 200000}};
set<string> department = {"Администрация", "Бухгалтерия", "Продажи"};
string selectPost();
```

map<string, int> post = {{"Генеральный директор", 500000}, ...};

- Предустановленный список должностей и их зарплат.

- Используется при вводе данных, чтобы пользователь выбирал из фиксированных значений.

```
map<string, int> post = {{"Генеральный директор", 500000},{"Главный бухгалтер", 400000}
set<string> department = {"Администрация", "Бухгалтерия", "Продажи"};
string selectPost();
```

-

set<string> department = {"Администрация", "Бухгалтерия", "Продажи"};

- Набор существующих отделов (подразделений).

- Пользователь может выбрать только из этого списка.

```
    SaveLoadData sld;
};
```

SaveLoadData sld;

- Объект для сохранения и загрузки данных. Используется в конструкторе/ деструкторе.

Вспомогательные методы:

- string selectPost()

Помогает пользователю выбрать должность из доступных. Возвращает строку с названием должности.

- string selectDepart()

Позволяет выбрать отдел из существующего списка.

- int enterServiceNumber()

Обеспечивает ввод уникального номера сотрудника. Используется для поиска, удаления или перевода.

- void printLines(vector<string> lines)

Универсальный метод для вывода нескольких строк текста на экран. Упрощает форматирование вывода.

Связь empdict.h с другими модулями

1. empdict.h использует:
   - saveloaddata.h — для структуры Employee и класса SaveLoadData;
   - стандартные контейнеры map, set, string.

2. empdict.h включается в главный файл kadry.cpp, где создаётся объект EmpDict и через него пользователь взаимодействует с системой.

Функциональное описание файла empdict.cpp:

Файл empdict.cpp содержит реализацию методов класса EmpDict, объявленного в empdict.h. В нём размещена основная бизнес-логика управления персоналом: приём, увольнение, учёт прогулов и вывод данных на печать. Модуль выполняет функции учёта сотрудников предприятия через интерфейс командной строки.

Основные компоненты:

Конструктор EmpDict::EmpDict()

При создании объекта класса EmpDict происходит загрузка базы сотрудников из файла:

```
EmpDict::EmpDict()
{
    empDict = sld.loadData();
}
```

empDict = sld.loadData();

Это гарантирует, что при запуске программы пользователь продолжит работу с уже сохранённой базой данных.

Основные методы управления:

1. addEmployee()

Позволяет добавить нового сотрудника:

- Вводятся фамилия, имя, отчество, возраст.
- Выбирается должность из списка (через selectPost()).
- Выбирается отдел (через selectDepart()).
- Генерируется уникальный табельный номер (enterServiceNumber()).

После добавления база сохраняется:

```
    emp.post = selectPost();
    emp.department = selectDepart();
    empDict.insert({enterServiceNumber(),emp});
    sld.saveData(empDict);
}
```

sld.saveData(empDict);

2. displayEmp()

Выводит таблицу всех сотрудников, которые не уволены.

Отображает: табельный номер, ФИО, возраст, должность, отдел, зарплату (находится по должности из post).

Затем предлагает 3 действия:

‘Y’ — отметить прогул. Добавляет сегодняшнюю дату в список прогулов сотрудника.

‘I’ — дополнительная информация. Пользователь выбирает: список уволенных; список отсутствующих; список присутствующих.

‘P’ — печать списка сотрудников. Создаётся временный файл и отправляется на принтер через PowerShell.

3. dismissalEmployee()

- Выводит всех сотрудников с их табельными номерами.
- Пользователь выбирает сотрудника, которого нужно уволить.
- Увольнение реализуется изменением поля isDismis = true.
- После этого база сохраняется.

4. moveToDepartment()

- Позволяет перевести сотрудника в другой отдел.
- Отображаются все сотрудники (кроме уволенных).
- Пользователь вводит табельный номер и выбирает новый отдел.
- После изменения отделов база сохраняется.

Вспомогательные методы

1. selectPost()

- Выводит нумерованный список должностей (например, "Менеджер", "Бухгалтер").
- Пользователь вводит номер из списка.
- Возвращает выбранную строку.

2. selectDepart()

- Аналогично selectPost(), но для отделов ("Продажи", "Бухгалтерия").
- Возвращает название отдела.

3. enterServiceNumber()

- Запрашивает ввод табельного номера.
- Если такой номер уже существует, просит ввести заново.
- Возвращает уникальный номер.

4. printLines(vector<string> lines)

- Принимает список строк (список сотрудников).
- Сохраняет их во временный файл temp_print.txt.
- С помощью PowerShell-скрипта отправляет содержимое на печать с заданным шрифтом.

Особенности реализации программы:

1. Программа работает в интерактивном режиме через консоль.
2. Все действия пользователя сохраняются после каждого изменения, что обеспечивает сохранность данных.
3. Программа отделяет бизнес-логику (добавление, увольнение) от хранения (реализовано в SaveLoadData).
4. Реализована печать на принтере, что расширяет практическое применение системы в реальной организации.

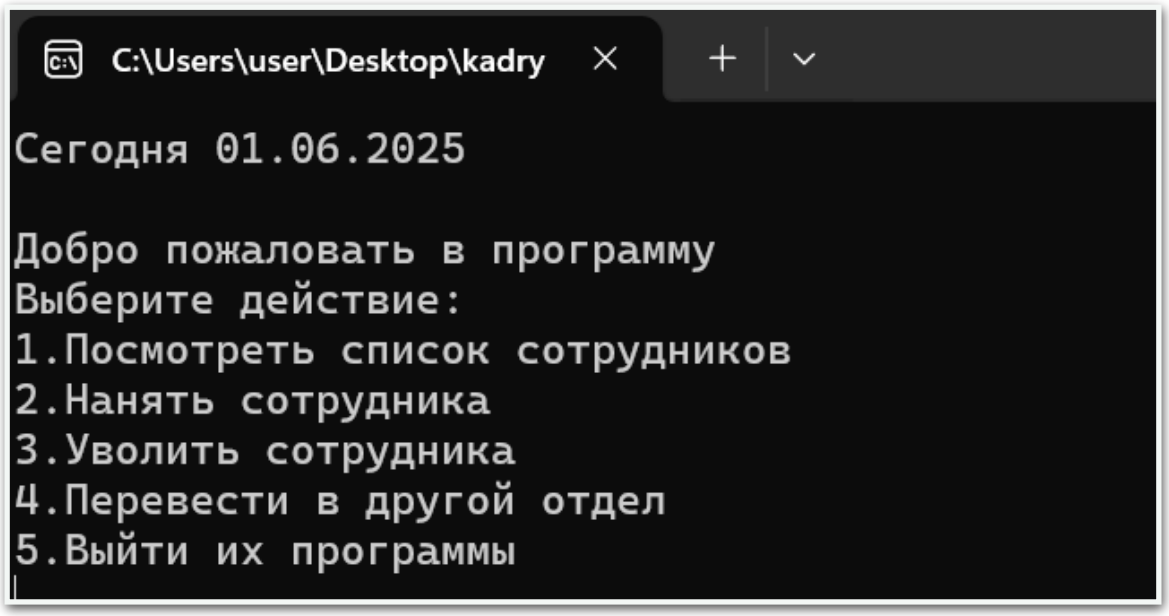
Инструкция по использованию

После запуска программы выводится главное окно с меню, содержащим пункты, соответствующие основным функциям учёта персонала: просмотр списка сотрудников, приём на работу, увольнение, перевод между отделами и завершение работы.

Для начала взаимодействия с системой рекомендуется выполнить поэтапное заполнение базы данных сотрудников, так как на основе этих сведений в дальнейшем выполняются основные операции кадрового учёта, включая фиксацию информации об отсутствии, переводах и формирование ведомостей для печати.

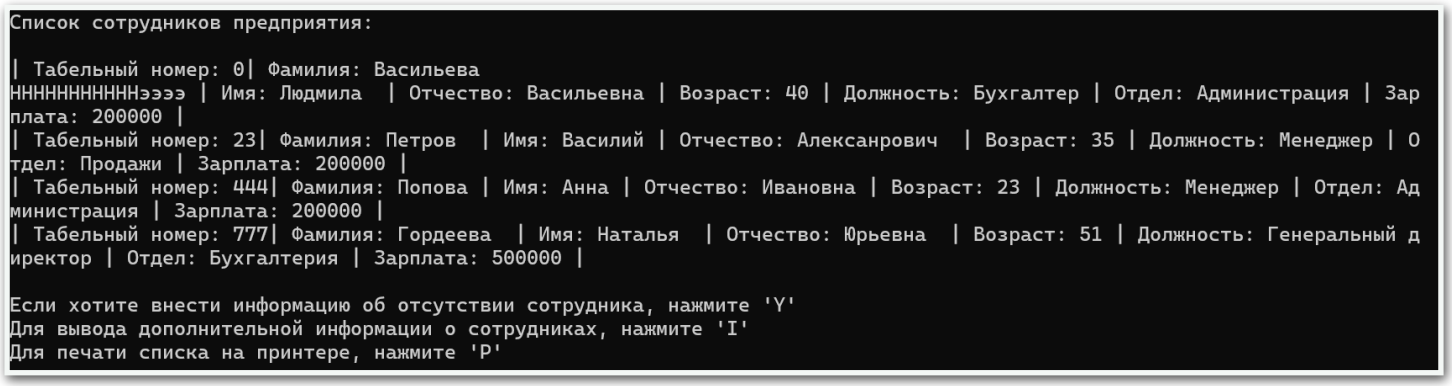
Для выбора необходимого действия требуется ввести номер соответствующего пункта меню (от 1 до 5) и подтвердить ввод. После выбора система поэтапно запрашивает нужные данные, сопровождая работу текстовыми подсказками. Интерфейс реализован в виде простого текстового взаимодействия, не требующего специальных технических навыков.

Для возврата в главное меню необходимо ввести 0 — будет выполнен переход в начальное окно программы.



В списке представлены следующие действия:

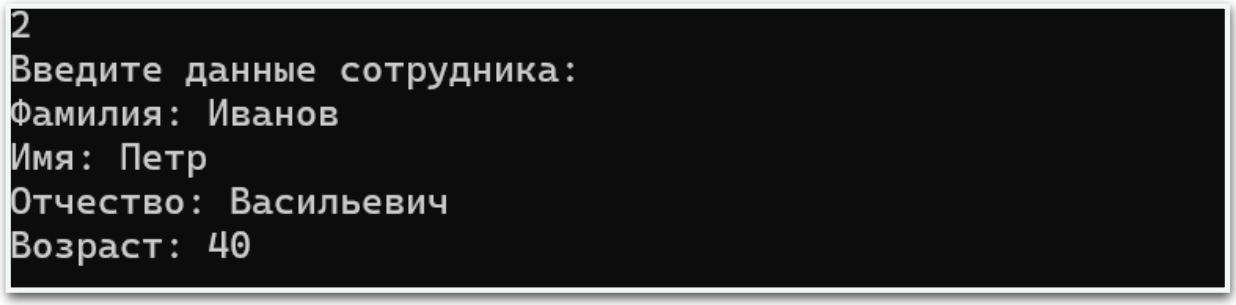
1. Посмотреть список сотрудников — выполняется вывод всех сотрудников, находящихся в штате. Дополнительно предоставляется возможность:



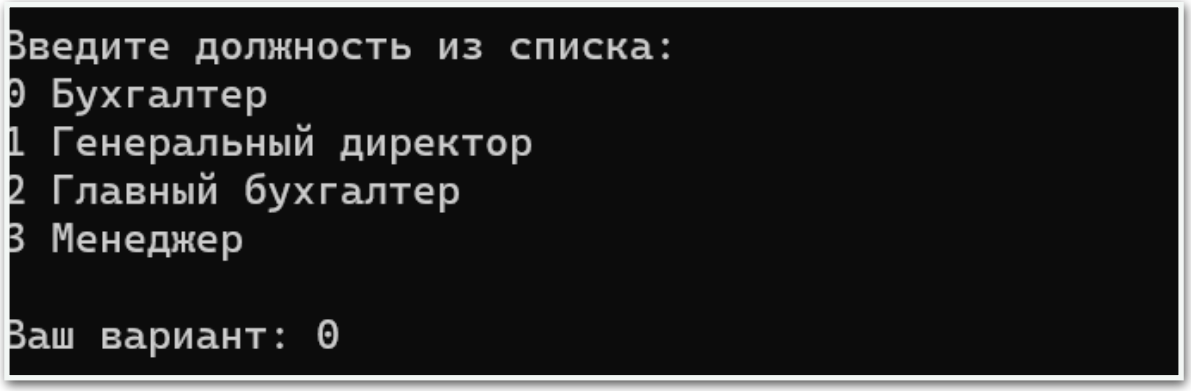
- Внести информацию об отсутствии сотрудникана текущую дату;
- Вывести дополнительную информацию о сотрудниках;
- Напечатать список на принтере.

2. Нанять сотрудника — инициируется последовательный ввод данных:

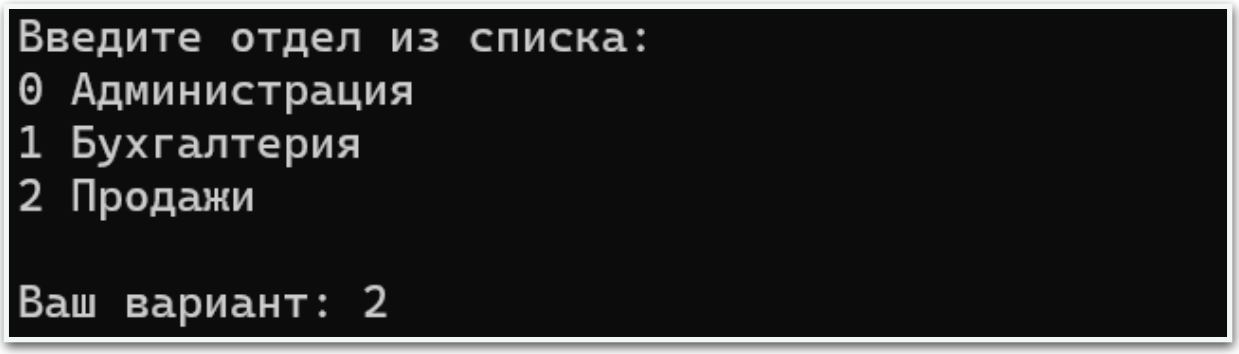
a) фамилии, имени, отчества, возраста



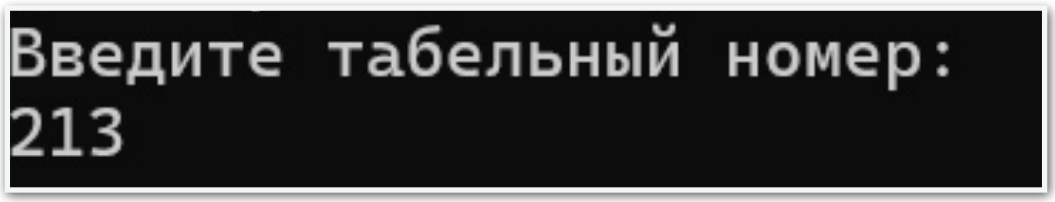
b) выбор должности, ожидается выбор необходимой цифры



c) выбор отдела из предложенного списка, аналогично необходим ввод цифры

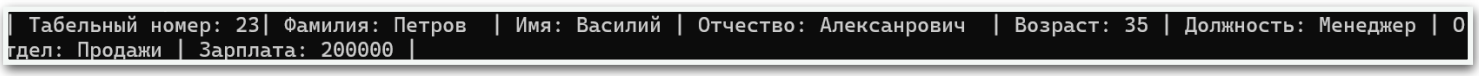


d) ввод уникального табельного номера

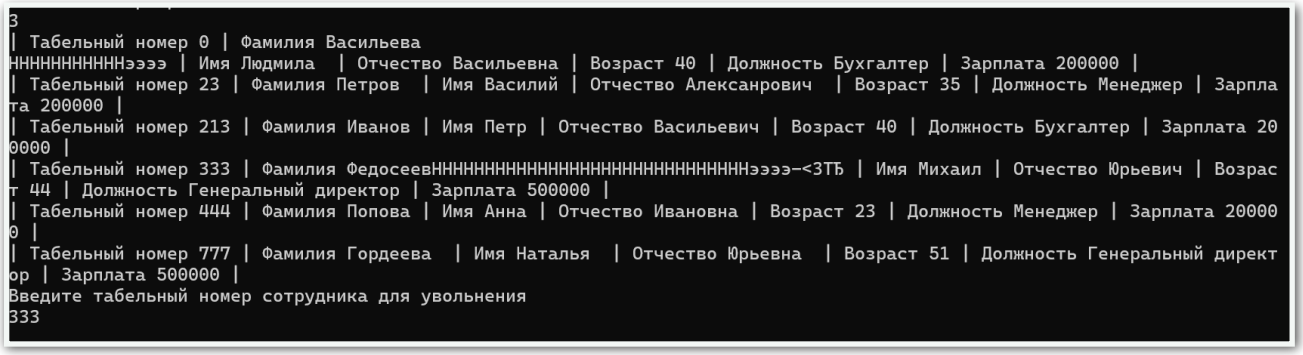


Каждой должности в программе заранее установлена фиксированная заработная плата. После выбора должности она автоматически связывается с сотрудником и отображается при просмотре списка работников.

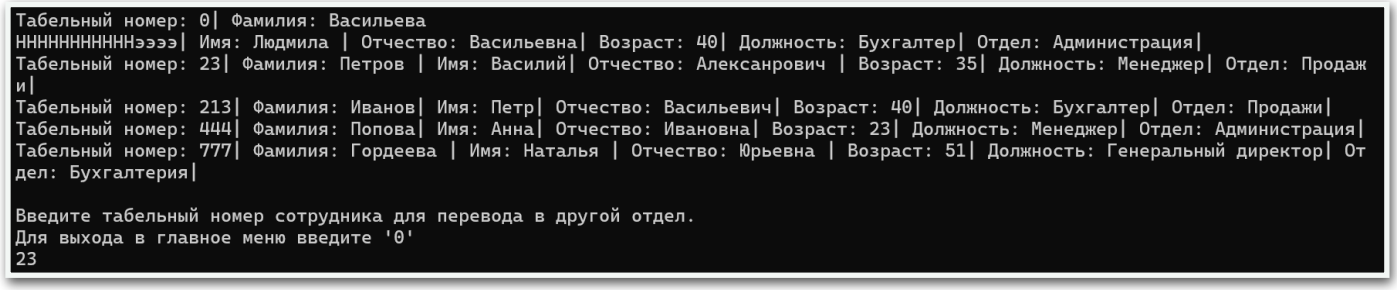
Рис. Окончательный вариант ввода всех данных по пункту 2 представлен в завершённой форме, включающей все необходимые параметры.



3. Уволить сотрудника — отображается список сотрудников, после чего осуществляется выбор нужного по табельному номеру. При подтверждении сотрудник отмечается как уволенный, исключается из основного списка, но сохраняется в базе данных.



4. Перевести в другой отдел — предоставляется возможность изменить подразделение сотрудника. Сначала выбирается табельный номер, затем



осуществляется выбор нового отдела.

5. Выйти из программы — завершает выполнение приложения. Все внесённые изменения автоматически сохраняются в файл, что обеспечивает сохранность данных при последующих запусках.

При возникновении ошибки требуется перезапуск программы.

Заключение

В ходе исследования рассмотрены различные аспекты автоматизации кадрового учёта на предприятии, включая особенности хранения информации о персонале, типовые операции отдела кадров, требования к структуре данных и возможные способы реализации с использованием языка программирования C++ и стандартной библиотеки шаблонов STL.

Основные результаты заключаются в следующем:

1) разработана и реализована консольная программа для автоматизации работы отдела кадров
2) создана структура для хранения информации о сотрудниках,
3) реализованы функции добавления, увольнения, перевода и учёта прогулов, обеспечено сохранение и загрузка данных с помощью файлового хранения
4) также реализован удобный пользовательский интерфейс в виде текстового меню, позволяющий легко взаимодействовать с системой без специальных технических навыков.

Заключение данной курсовой работы позволило достичь поставленных целей и задач исследования. Разработанное консольное приложение демонстрирует работу простой, но функциональной системы учёта сотрудников, позволяющей автоматизировать основные кадровые процессы. Решение обладает понятным интерфейсом, не требует установки дополнительных компонентов и может быть использовано в небольших организациях для упрощения рутинной кадровой работы.

Список использованных ресурсов

1. Страуструп Б. Язык программирования C++. Специальное издание. — М.: Вильямс, 2014.

2. Молчанов П. А. Программирование на C++ для начинающих. — М.: Наука и Техника, 2021.

3. cppreference.com — Электронный справочник по стандартной библиотеке C++ [URL: https://en.cppreference.com]

4. Microsoft Docs (MSDN). Документация по работе с консолью Windows и библиотекой <Windows.h> [URL: https://learn.microsoft.com]

5. https://en.cppreference.com/w/cpp/container/map/ - Использовался для изучения особенностей контейнера map из STL, применяемого для хранения базы сотрудников.

Приложение

kadry.cpp

```
#include <iostream>
#include <string>
#include <locale>
#include <Windows.h>
#include "empdict.h"

using namespace std::chrono;

int main()
{
   SetConsoleOutputCP(1251);
   SetConsoleCP(1251);

   EmpDict eDict;
   while (true)
   {
      int action{};
      system("cls");
      auto today = year_month_day{ floor<days>(system_clock::now()) };
                  std::cout << "Сегодня " << std::format("{:%d.%m.%Y}\n",
sys_days{today}) << endl;
      cout << "Добро пожаловать в программу" << endl;
      cout << "Выберите действие:" << endl;
      cout << "1.Посмотреть список сотрудников" << endl;
      cout << "2.Нанять сотрудника" << endl;
      cout << "3.Уволить сотрудника" << endl;
      cout << "4.Перевести в другой отдел" << endl;
      cout << "5.Выйти их программы" << endl;
```

```
            wcin >> action;
            switch (action)
            {
            case 1:
                cin.ignore();
                eDict.displayEmp();
                cin.get();
                continue;
                break;
            case 2:

                cin.ignore();
                eDict.addEmployee();
                break;
            case 3:
                eDict.dismissalEmployee();
                break;
            case 4:
                eDict.moveToDepartment();
                break;
            case 5:
                return 0;
                break;
            default:
                    cout << "Введён не корректный идентификатор пункта меню" <<
endl;
                break;
            }
        }

        return 0;
```

```
}
```

saveloaddata.h

```
#ifndef SAVELOADDATA_H
#define SAVELOADDATA_H

#include <string>
#include <map>
#include <chrono>
#include <vector>

using namespace std;
using namespace chrono;

struct Employee{
    int number{};

    string firstName{};
    string lastName{};
    string patronimic{};

    int     age{};
    string  post{};
    string  department{};

    bool isDismis = false;
    vector<year_month_day> vTruancy{};
};

class SaveLoadData
```

```
{
public:
    SaveLoadData();

    bool saveData(map<int, Employee> empDict);

    map<int, Employee>  loadData();

private:

};

#endif // SAVELOADDATA_H
```

saveloaddata.cpp

```
#include "saveloaddata.h"
#include <fstream>
#include <iostream>
#include <format>

SaveLoadData::SaveLoadData()
{

}

bool SaveLoadData::saveData(map<int, Employee> empDict)
{
    ofstream file("database.db", ios::binary);
    if(!file)
    {
```

```
            cout << "Ошибка открытия файла" << endl;
            return false;
        }

        size_t size = empDict.size();
        file.write(reinterpret_cast<const char*>(&size),  sizeof(size));
        for(const auto &pair:empDict)
        {
            file.write(reinterpret_cast<const char*>(&pair.first), sizeof(pair.first));

            // Записываем длину строки и саму строку
            Employee tmp = pair.second;

                                file.write(reinterpret_cast<const   char*>(&tmp.number),
sizeof(tmp.number))     ;

            size_t fnSize = tmp.firstName.size()*2;
            file.write(reinterpret_cast<const char*>(&fnSize),    sizeof(size_t))  ;
                        file.write(reinterpret_cast<const  char*>(tmp.firstName.c_str()),
fnSize)  ;

            size_t lnSize = tmp.lastName.size();
            file.write(reinterpret_cast<const char*>(&lnSize),    sizeof(size_t))  ;
             file.write(reinterpret_cast<const char*>(tmp.lastName.c_str()),      lnSize)
;

            size_t pSize = tmp.patronimic.size();
            file.write(reinterpret_cast<const char*>(&pSize),    sizeof(size_t))  ;
            file.write(reinterpret_cast<const char*>(tmp.patronimic.c_str()),   pSize) ;
```

```
                    file.write(reinterpret_cast<const   char*>(&tmp.age),
sizeof(tmp.age))       ;

        size_t postSize = tmp.post.size();
        file.write(reinterpret_cast<const char*>(&postSize),    sizeof(size_t))  ;
         file.write(reinterpret_cast<const char*>(tmp.post.c_str()),          postSize)
;

        size_t dSize = tmp.department.size();
        file.write(reinterpret_cast<const char*>(&dSize),    sizeof(size_t))  ;
        file.write(reinterpret_cast<const char*>(tmp.department.c_str()),   dSize) ;

        file.write(reinterpret_cast<const char*>(&tmp.isDismis), 1);

        size_t tSize = tmp.vTruancy.size();
        file.write(reinterpret_cast<const char*>(&tSize), sizeof(tSize));

        for (const year_month_day date : tmp.vTruancy)
        {
           int day = static_cast<unsigned>(date.day());
           int month = static_cast<unsigned>(date.month());
           int year = static_cast<int>(date.year());
           file.write(reinterpret_cast<const char*>(&day), 4);
           file.write(reinterpret_cast<const char*>(&month), 4);
           file.write(reinterpret_cast<const char*>(&year), 4);
        }
    }

    file.flush();
    file.close();
    return true;
```

```
}

map<int, Employee> SaveLoadData::loadData()
{
   map<int, Employee> loadMap{};
   ifstream file("database.db", ios::binary);
   if(!file)
   {
      cout << "Ошибка открытия файла" << endl;
      return loadMap;
   }
   size_t map_size{};
   file.read(reinterpret_cast<char*>(&map_size), sizeof(map_size));
   for (size_t i = 0; i < map_size; ++i) {
      // Читаем ключ
      int key{};
      file.read(reinterpret_cast<char*>(&key), sizeof(key));
      Employee tmp;
       file.read(reinterpret_cast<char*>(&tmp.number),      sizeof(tmp.number))
;

      size_t fnSize{};
      file.read(reinterpret_cast<char*>(&fnSize),   sizeof(size_t))  ;
      tmp.firstName.resize(fnSize);
      file.read(reinterpret_cast<char*>(&tmp.firstName[0]),   fnSize)  ;

      size_t lnSize{};
      file.read(reinterpret_cast<char*>(&lnSize),   sizeof(size_t))  ;
      tmp.lastName.resize(lnSize);
      file.read(reinterpret_cast<char*>(&tmp.lastName[0]),    lnSize)   ;
```

```
        size_t pSize{};
        file.read(reinterpret_cast<char*>(&pSize),   sizeof(size_t))  ;
        tmp.patronimic.resize(pSize);
        file.read(reinterpret_cast<char*>(&tmp.patronimic[0]),  pSize) ;


        file.read(reinterpret_cast<char*>(&tmp.age),        sizeof(tmp.age))       ;

        size_t postSize{};
        file.read(reinterpret_cast<char*>(&postSize),   sizeof(size_t))  ;
        tmp.post.resize(postSize);
        file.read(reinterpret_cast<char*>(&tmp.post[0]),       postSize)      ;

        size_t depSize{};
        file.read(reinterpret_cast<char*>(&depSize),   sizeof(size_t))  ;
        tmp.department.resize(depSize);
        file.read(reinterpret_cast<char*>(&tmp.department[0]),  depSize) ;



        file.read(reinterpret_cast<char*>(&tmp.isDismis), sizeof(tmp.isDismis));

        size_t dateMap_size{};
                        file.read(reinterpret_cast<char*>(&dateMap_size),
sizeof(dateMap_size));
        for (int dateIndex = 0; dateIndex < dateMap_size; dateIndex++)
        {
           int cday{};
           int cmonth{};
           int cyear{};
           file.read(reinterpret_cast<char*>(&cday), 4);
```

```
            file.read(reinterpret_cast<char*>(&cmonth), 4);
            file.read(reinterpret_cast<char*>(&cyear), 4);
                                year_month_day  tmpDate(static_cast<year>(cyear),
static_cast<month>(cmonth), static_cast<day>(cday));

            tmp.vTruancy.push_back(tmpDate);
        }
        pair<int, Employee> p(key, tmp);
        loadMap.insert(p);
    }
    file.close();

    return loadMap;
}
```

empdict.h

```
#ifndef EMPDICT_H
#define EMPDICT_H

#include <map>
#include <set>
#include <string>
#include "saveloaddata.h"

using namespace std;

class EmpDict
{
public:
    EmpDict();
```

```
    void addEmployee();

    void dismissalEmployee();

    void moveToDepartment();

    void displayEmp();


    ~EmpDict();

private:
    map<int, Employee> empDict{};

     map<string, int> post = {{"Генеральный директор", 500000},{"Главный бухгалтер", 400000},{"Бухгалтер", 200000},{"Менеджер", 200000}};
          set<string> department = {"Администрация", "Бухгалтерия", "Продажи"};
    string selectPost();

    string selectDepart();
    int enterServiceNumber();
    void printLines(vector<string> lines);

    SaveLoadData sld;
};


#endif // EMPDICT_H
```

empdict.cpp

```
#include "empdict.h"
#include <iostream>
#include <codecvt>
#include <fstream>

EmpDict::EmpDict()
{
   empDict = sld.loadData();
}


void EmpDict::addEmployee()
{
   Employee emp;
   cout << "Введите данные сотрудника: " << endl;
   cout << "Фамилия: ";
   getline(cin, emp.firstName);
   cout << "Имя: ";
   getline(cin, emp.lastName);
   cout << "Отчество: ";
   getline(cin, emp.patronimic);
   cout << "Возраст: ";
   cin >> emp.age;

   emp.post = selectPost();
   emp.department = selectDepart();
   empDict.insert({enterServiceNumber(),emp});
   sld.saveData(empDict);
}
```

```
void EmpDict::displayEmp()
{
   system("cls");
   cout << "Список сотрудников предприятия:" << endl;
   cout << endl;
   if (empDict.size() > 0)
   {
      for (auto empPair : empDict)
      {
         if (empPair.first >= 0 && !empPair.second.isDismis)
         {
            Employee tmp = empPair.second;
              cout << "| Табельный номер: " << empPair.first <<  "| Фамилия: "
<< tmp.firstName << " | Имя: " << tmp.lastName << " | Отчество: " <<
tmp.patronimic << " | Возраст: "
                  << tmp.age << " | Должность: " << tmp.post << " | Отдел: "  <<
tmp.department << " | Зарплата: " << post.at(tmp.post) << " | " << endl;
         }
      }
      char isTruancy{};
      cout << endl;
       cout << "Если хотите внести информацию об отсутствии сотрудника,
нажмите 'Y'" << endl;
          cout << "Для вывода дополнительной информации о сотрудниках,
нажмите 'I'" << endl;
      cout << "Для печати списка на принтере, нажмите 'P'" << endl;
      cin >> isTruancy;
      if (isTruancy == 'P')
      {
         vector<string> lines;
```

```
                lines.push_back("Список сотрудников в штате\n");
                for (auto empPair : empDict)
                {
                    if (empPair.first >= 0 && !empPair.second.isDismis)
                    {
                        Employee tmp = empPair.second;

                                                string printLine = "| Табельный номер: " +
to_string(empPair.first) + "| Фамилия: " + tmp.firstName + " | Имя: " +
tmp.lastName + " | Отчество: " + tmp.patronimic + " | Должность " + tmp.post + " |
\n ";
                        lines.push_back(printLine);
                    }
                }
                printLines(lines);
            }
            else if (isTruancy == 'Y')
            {
                int empNum{};
                 cout << "Введите табельный номер отсутствующего сотрудника из
таблицы:" << endl;
                cin >> empNum;
                auto today = year_month_day{ floor<days>(system_clock::now()) };
                empDict[empNum].vTruancy.push_back(today);
            }
            else if (isTruancy == 'I')
            {
                int punktNum{};

                while (true)
                {
```

```
system("cls");
    cout << "Какую дополнительную информацию вы хотели бы получить?" << endl;
cout << "1. Список уволенных сотрудников." << endl;
    cout << "2. Список сотрудников, отсутствующих на рабочем месте" << endl;
    cout << "3. Список сотрудников, присутствующих на рабочем месте" << endl;
cout << "4. Вернуться в главное меню" << endl;
cin >> punktNum;
auto today = year_month_day{ floor<days>(system_clock::now()) };
system("cls");
switch (punktNum)
{
case 1:
   cout << "Список уволеных сотрудников:" << endl;
   cout << endl;
   for (const auto empPair : empDict)
   {
      if (empPair.second.isDismis)
      {
         Employee tmp = empPair.second;
                cout << "| Табельный номер: " << empPair.first << "| Фамилия: " << tmp.firstName << " | Имя: " << tmp.lastName << " | Отчество: " << tmp.patronimic << " | Возраст: "
                    << tmp.age << " | Должность: " << tmp.post << " | Зарплата: " << post.at(tmp.post) << " | " << endl;
      }
   }
   cin.ignore();
   cin.get();
```

```
                break;
            case 2:
                cout << "Список отсутствующих сотрудников:" << endl;
                cout << endl;
                for (const auto empPair : empDict)
                {
                                    if (find(empPair.second.vTruancy.begin(),
empPair.second.vTruancy.end(), today )!= empPair.second.vTruancy.end() && !
empPair.second.isDismis)
                    {
                        Employee tmp = empPair.second;
                                cout << "| Табельный номер: " << empPair.first << "|
Фамилия: " << tmp.firstName << " | Имя: " << tmp.lastName << " | Отчество: " <<
tmp.patronimic << " | Возраст: "
                                    << tmp.age << " | Должность: " << tmp.post << " |
Зарплата: " << post.at(tmp.post) << " | " << endl;
                    }
                }
                cin.ignore();
                cin.get();
                break;
            case 3:
                cout << "Список присутствующих сотрудников:" << endl;
                cout << endl;
                for (const auto empPair : empDict)
                {
                                    if (find(empPair.second.vTruancy.begin(),
empPair.second.vTruancy.end(), today) == empPair.second.vTruancy.end() && !
empPair.second.isDismis)
                    {
                        Employee tmp = empPair.second;
```

```
                                cout << "| Табельный номер: " << empPair.first << "|
Фамилия: " << tmp.firstName << " | Имя: " << tmp.lastName << " | Отчество: " <<
tmp.patronimic << " | Возраст: "
                                    << tmp.age << " | Должность: " << tmp.post << " |
Зарплата: " << post.at(tmp.post) << " | " << endl;
                            }
                        }
                        cin.ignore();
                        cin.get();
                        break;
                    case 4:
                        return;
                        break;
                    default:
                        break;
                    }
                }
            }
        }
        else
        {
            system("cls");
            cout << "Сотрудники отсутствуют. Наймите кого ни будь" << endl;
        }
    }

    EmpDict::~EmpDict()
    {
        sld.saveData(empDict);
    }
```

```
void EmpDict::dismissalEmployee()
{
   if(empDict.size() > 0)
   {
      for(auto empPair:empDict)
      {
         Employee tmp = empPair.second;
           cout << "| Табельный номер " << empPair.first << " | Фамилия " << tmp.firstName << " | Имя " << tmp.lastName << " | Отчество " << tmp.patronimic << " | Возраст " << tmp.age << " | Должность " << tmp.post << " | Зарплата " << post.at(tmp.post) << " |" << endl;
      }
      int num{};
        cout << "Введите табельный номер сотрудника для увольнения" << endl;
      cin >> num;
      Employee *tmpE = &empDict[num];
      tmpE->isDismis = true;

      sld.saveData(empDict);
   }
   else
   {
      cout << "Сотрудники отсутствуют. Наймите кого ни будь" << endl;
   }
}

void EmpDict::moveToDepartment()
{
   system("cls");
```

```
if(empDict.size() > 0)
{
   while (true)
   {
      system("cls");
      for(auto empPair:empDict)
      {
         if (empPair.first >= 0 && !empPair.second.isDismis)
         {
            Employee tmp = empPair.second;
              cout << "Табельный номер: " << empPair.first << "| Фамилия: "
<< tmp.firstName << "| Имя: " << tmp.lastName << "| Отчество: " <<
tmp.patronimic <<
                   "| Возраст: " << tmp.age << "| Должность: " << tmp.post << "|
Отдел: " << tmp.department << "|" << endl;
         }
      }
      int num{};
      cout << endl;
          cout << "Введите табельный номер сотрудника для перевода в
другой отдел." << endl;
      cout << "Для выхода в главное меню введите '0'" << endl;
      cin >> num;
      if (num == 0)
         break;
      if (!empDict.contains(num))
      {
         cout << "Не верный табельный номер, повторите ввод!" << endl;
         cin.ignore();
         cin.get();
         continue;
```

```
        }
        cout << "Выберите отдел." << endl;
        cin.ignore();
        int selPost{};

        int counter = 1;
        for(const auto pDep: department)
        {
           cout << counter << ". "<< pDep << endl;
           counter++;
        }
        cin >> selPost;

        counter = 1;
        for (const auto pDep : department)
        {
           if(counter == selPost)
              empDict.at(num).department = pDep;
           counter++;
        }

        sld.saveData(empDict);
        break;
     }

  }
  else
  {
     cout << "Сотрудники отсутствуют. Наймите кого ни будь" << endl;
  }
}
```

```
string EmpDict::selectPost()
{
    string returnPost{};
    cin.ignore();
    while(true)
    {

        cout << endl;
        cout << "Введите должность из списка:" << endl;
        int counter{};
        int postNum{};
        for(pair<string, int> pPost: post)
        {
            cout << counter << " "<< pPost.first << endl;
            counter++;
        }
        cout << endl;
        cout << "Ваш вариант: ";
        cin >> postNum;
        counter = 0;

        for (pair<string, int> pPost : post)
        {
            if(counter == postNum)
            {
                returnPost = pPost.first;
                break;
            }
            counter++;
        }
```

```
        if(returnPost.empty())
        {
            cout << "Вы ввели не верный вариант должности" << endl;
        }
        else
        {
            break;
        }
    }
    return returnPost;
}

string EmpDict::selectDepart()
{
    string returnDep{};
    while(true)
    {
        cout << "Введите отдел из списка:" << endl;
        int counter{};
        int postNum{};
        for(string pDep: department)
        {
            cout << counter << " " << pDep << endl;
            counter++;
        }
        cout << endl;
        cout << "Ваш вариант: ";
        cin >> postNum;
        counter = 0;
        for (string pDep : department)
```

```
        {
            if (counter == postNum)
            {
                returnDep = pDep;
                break;
            }
            counter++;
        }

        if(returnDep.empty())
        {
            cout << "Вы ввели не верный отдела" << endl;
        }
        else {
            break;
        }
    }
    return returnDep;
}

int EmpDict::enterServiceNumber()
{
    int number{};
    while(true)
    {
        cout << "Введите табельный номер:" << endl;
        cin >> number;
        if(empDict.find(number) != empDict.end())
        {
            cout << "Такой номер уже есть" << endl;
        }
```

```
            else
            {
                break;
            }
        }
        return number;
    }

    void EmpDict::printLines(vector<string> lines)
    {
        // Создаем временный файл с содержимым для печати

        std::ofstream out("temp_print.txt");
        for (const auto line : lines)
        {
            out << line;

        }
        out.close();


        // PowerShell команда для печати с маленьким шрифтом
        std::string command = "powershell -Command \""
            "$content = Get-Content 'temp_print.txt' -Raw; "
            "Add-Type -AssemblyName System.Drawing; "
            "$font = New-Object System.Drawing.Font('Courier New', 8); "
            "$pd = New-Object System.Drawing.Printing.PrintDocument; "
                                "$pd.PrinterSettings = New-Object
System.Drawing.Printing.PrinterSettings; "
            "$pd.Add_PrintPage({ "
```

```
                    "          p a r a m ( [ o b j e c t ] $ s e n d e r ,
[System.Drawing.Printing.PrintPageEventArgs]$e); "
                    "          $e.Graphics.DrawString($content,  $font,
[System.Drawing.Brushes]::Black, 10, 10); "
            "}); "
            "$pd.Print(); \"";

        system(command.c_str());
    }
```