МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГТУ», ВГТУ)

Факультет информационных технологий и компьютерной безопасности

(факультет)

Кафедра систем управления и информационных технологий в строительстве

КУРСОВОЙ ПРОЕКТ

по дисциплине: Основы программирования и алгоритмизации

Тема: Разработка файловой базы данных «Заводы» ⠀

Расчетно-пояснительная записка

Разработал студент Д.А. Чуфицкий

Подпись, дата Инициалы, фамилия

Руководитель О.В. Курипта

Подпись, дата Инициалы, фамилия

Члены комиссии

Подпись, дата Инициалы, фамилия

Подпись, дата Инициалы, фамилия

Подпись, дата Инициалы, фамилия

Нормоконтролер

Подпись, дата Инициалы, фамилия

Защищена Оценка

дата

Воронеж

2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГТУ», ВГТУ)

Факультет информационных технологий и компьютерной безопасности

(факультет)

Кафедра систем управления и информационных технологий в строительстве

ЗАДАНИЕ

на курсовой проект

по дисциплине: Основы программирования и алгоритмизации

Тема: Разработка файловой базы данных «Заводы»

Студент бИД-241 Чуфицкий Денис Андреевич

Группа, фамилия, имя, отчество

Номер варианта

Технические условия CPU: Intel Core i3-12100F GPU: Nvidia GeForce GTX 1650 ОС: Windows 10 64-раз ОЗУ: 16 ГБ ⠀

Содержание и объем работы (графические работы, расчеты и прочее): страниц: , рисунков , таблиц:

Сроки выполнение этапов:

Сроки защиты курсового проекта:

Руководитель О.В. Курипта

Подпись, дата Инициалы, фамилия

Задание принял студент Д.А. Чуфицкий

Подпись, дата Инициалы, фамилия

Замечание руководителя

Содержание

ВВЕДЕНИЕ

В современном мире информация является одним из самых ценных ресурсов, и эффективное управление данными становится ключевым аспектом успешной деятельности организаций. Базы данных представляют собой структурированные системы, которые позволяют хранить, обрабатывать и извлекать информацию с высокой степенью эффективности.

В рамках данного курсового проекта будет рассмотрена методология проектирования и реализации базы данных “Заводы” на языке программирования С, которая позволит систематизировать и хранить информацию о заводах и их работниках. База данных позволит не только упорядочить данные, но и обеспечить их легкий доступ и обработку для дальнейшего анализа.

 База данных будет включать такие ключевые поля, как номер завода, количество персонала, которые будут уникальным идентификатором для каждого предприятия, а также информацию о персонале завода, представленную в виде вложенной структуры. В частности, для каждого работника будут храниться фамилия, имя и отчество, что позволит точно идентифицировать каждого сотрудника.

Кроме того, база данных будет содержать такие важные характеристики, как возраст работников, специальность, средний оклад и стаж. Возраст позволит анализировать демографическую структуру персонала, а специальность — учитывать квалификацию работников, что важно для планирования кадровой политики. Средний оклад будет рассчитываться на основе данных о зарплатах сотрудников и поможет в оценке финансовых затрат на персонал. Стаж, представленный в годах, даст возможность оценить опыт работников, что также является важным фактором при принятии управленческих решений.

Для достижения поставленной цели, необходимо решить следующие задачи:

1) предложить структуру для описания отдельных полей, включающую не менее пяти полей, двух и более различных типов данных. Название и тип полей определяется самостоятельно по варианту.

2) Организовать запись и чтение данных (отдельных записей базы данных) из файла, обеспечив корректное форматирование данных для удобства их последующей обработки и минимизируя риск потери информации при чтении и записи.

3) Создать функцию для поиска записи по указанным пользователем значениям полей, при этом обеспечивая легкость и удобство ввода искомых данных, чтобы пользователи могли быстро и без затруднений находить нужную информацию.

4) предусмотреть возможность упорядочивания записей по номерам и по среднему окладу заводов для комфортного отображения данных в нужной последовательности.

5\*) обеспечить возможность добавления новых записей в файл базы данных, изменения загруженных и сохранения результатов по желанию пользователя.

Интерфейс программы будет разработан с акцентом на простоту и функциональность, что позволит пользователям легко выполнять необходимые действия. Он будет включать в себя четкие и понятные элементы управления, такие как кнопки и выпадающие списки, а также визуальные подсказки, которые помогут в навигации. Пользователи смогут легко ориентироваться в приложении, быстро находить нужные функции и эффективно взаимодействовать с данными, что сделает процесс работы более комфортным и продуктивным.

Из этого следует, что моя файловая база данных станет отличным средством для обеспечения хранения, форматирования, сортировки и, в целом, использованием информации для управления и анализа данных.

1. **Постановка задачи**

Разработать файловую базу данных “Заводы”

Для заданной базы данных:

1. Предложить структуру для описания отдельных полей (не менее пяти полей, двух и более различных типов данных); Название и тип полей: номер завода, персонал завода: фамилия, имя, отчество, средний возраст, специальность, оклад, стаж.
2. Организовать запись и чтение данных (отдельных записей базы данных) из файла;
3. Разработать функцию поиска записи по: По кол-ву работников, по специальности, по среднему стажу.
4. Предусмотреть возможность упорядочивания записей по: номерам заводов, по средним окладам заводов.
5. Обеспечить возможность добавления новых записей в файл базы данных, изменения загруженных и сохранения результатов по желанию пользователя.

Интерфейс программы должен обеспечивать следующие возможности:

1. Загрузка данных из заданного пользователем файла.
2. Вывод исходных данных из файла, результатов поиска и упорядочивания на экран.
3. Вывод диагностических сообщений в ходе проверки корректности ввода данных.
4. Задание пользователем значения одного из двух полей (ключей поиска) (\* или нескольких одновременно) для выполнения поиска.
5. Возможность выбора различных операций с записями.

При разработке файловой базы данных "Заводы" необходимо обеспечить компактность и структурированность данных, а также эффективное взаимодействие с файлами. Для этого следует использовать структуры, в которых каждый завод будет представлен отдельной записью с заранее определенными полями. В данном случае я планирую применять вложенные структуры для описания персонала завода, что позволит более удобно и компактно организовать хранение данных. Каждое поле будет отражать отдельный атрибут завода: номер, средний оклад, количество сотрудников, а также информацию о персонале, включая фамилию, имя, отчество, возраст, специальность, оклад и стаж.

Таблица 1.1 – структура завода

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| Номер завода | int | Поле для хранения номера завода. Тип int позволяет хранить информацию как целое число. |
| Количество  персонала | int | Поле для хранения количества персонала, что очень важно для поочередной записи и чтения информации. |
| Средний оклад | float | Поле для хранения среднего оклада всего завода, которое высчитывается отдельной функцией. Тип float позволяет точно отображать денежные значения. Важно для сортировки завода. |
| Работник | struct | Поле представляет собой структуру, содержащую информацию о каждом сотруднике завода. |

Таблица 1.2 – структура работника

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| Фамилия | char[] | Строковое поле типа char[] позволяет хранить строковые данные фиксированной длины, что упрощает запись и чтение из файла. Поле содержит фамилию работника. |
| Имя | char[] | Строковое поле содержит имя работника. |
| Отчество | char[] | Строковое поле содержит отчество работника. |
| Возраст | int | Поле для хранения возраста работника, возраст позволит анализировать демографическую структуру персонала. |
| Специальность | char[] | Строкове поле содержит название специальности рабочего. |
| Заработная плата | float | Поле для хранения оклада рабочего, важно для вычисления среднего оклада завода. |
| Опыт(стаж) | int | Поле для хранения стажа. |

В языке С **структура данных** будет организована с помощью struct, при этом каждое поле записи будет представлено отдельным элементом этой структуры. В моем случае следует применять вложенные структуры для описания персонала завода, что позволит более удобно и компактно организовать хранение данных.

Ключевые поля, которые будут применяться в функциях **поиска и сортировки**:

1. Поиск: по количеству работников, по специальности, по среднему стажу.
2. Сортировка: по номерам заводов, по средним окладам заводов.

Система управления базами данных должна гарантировать сохранение информации о заводе и его сотрудниках. Для этого требуется решить несколько задач, каждая из которых сосредоточена на **реализации** определенного **функционала**:

Таблица 1.3 – описание поставленных задач

|  |  |
| --- | --- |
| Задача | Описание |
| Предложить структуру | Создание структуры, представляющей запись о заводе, включающую в себя: номер завода, количество персонала, а также информацию о персонале завода, представленную в виде вложенной структуры. |
| Организовать функции для записи и чтения файла. | Разработка функции для записи и чтения записей из файла. Функция должна поддерживать возможность восстановления информации из файла. |
| Реализовать функцию поиска записи. | Функция поиска по количеству работников, специальности, среднему стажу. |
| Создание функции сортировки. | Функция сортировки по номерам заводов, среднему окладу. |
| Обеспечение возможности добавления новых записей и редактирования старых. | Функции для добавления новых записей, редактирования существующих. Рациональное использование указателей позволяет существенно сократить затраты как ресурсов, так и времени. |
| Возможность выбора различных операций | Создание удобного и хорошо организованного интерфейса. |
| Вывод диагностических сообщений в ходе проверок | Проверка каждой функции на отсутствие файлов, их переполнение. Помогает для выявления ошибок программы. |

1. **Конструирование программы**
   1. **Поэтапное описание программы**

Для организации и хранения информации для базы данных “Заводы” требуется создание двух структур данных, где первая структура будет содержать данные/характеристики рабочих заводов, а вторая структура будет являться самим заводов, в которой будут поля именно завода. В языке Си есть отдельная функция описания структуры “struct”, которая позволяет объединить в одной записи несколько различных типов данных.

Описание полей структуры **person**:

struct person {

char fam[20];

char im[20];

char ot[20];

int sr\_vrst;

char specialty[50];

float zp;

int opit;

};

Фамилия (fam) – тип char[20] выбран для хранения фамилии, так как фамилия обычно не превышает 20 символов. Использование массива символов позволяет хранить строку фиксированной длины, что удобно для работы с текстовыми данными.

Имя (im) – аналогично фамилии, тип char[20] выбран для хранения имени работника. Это позволяет хранить имя в пределах разумной длины, обеспечивая при этом простоту работы с текстом.

Отчество (ot) – тип char[20] также выбран для хранения отчества. Как и в предыдущих случаях, это позволяет хранить строку фиксированной длины, что удобно для формального обращения к работнику.

Средний возраст (sr\_vrst) – тип int выбран для хранения среднего возраста, так как возраст представляется целым числом. Использование целочисленного типа позволяет легко выполнять арифметические операции и сравнения.

Специальность (specialty) – тип char[50] выбран для хранения специальности работника, так как названия специальностей могут быть длиннее, чем фамилия или имя. Массив символов фиксированной длины позволяет хранить более длинные строки, что важно для точного описания квалификации работника.

Оклад (zp) – тип float выбран для хранения оклада, так как оклад может быть дробным числом (например, 25000.50). Использование типа с плавающей запятой позволяет точно представлять денежные значения и выполнять арифметические операции с ними.

Стаж (opit) – тип int выбран для хранения стажа в годах, так как стаж также представляется целым числом. Это позволяет легко выполнять операции, такие как сложение или сравнение стажа работников.

Описание полей структуры **factory**:

struct factory{

int naz\_factory;

struct person persons[MAX];

int col\_person;

float sr\_oklad;

};

Номер завода (naz\_factory) – тип int выбран для хранения номера завода, так как номер завода обычно представляется целым числом. Использование целочисленного типа позволяет легко выполнять операции сравнения и сортировки, а также обеспечивает компактное хранение данных.

Список работников (persons) – тип struct person persons[MAX] выбран для хранения массива работников завода. Использование массива структур позволяет удобно организовать данные о каждом работнике, обеспечивая доступ к информации о них по индексу. Максимальное количество работников задается константой MAX, что позволяет ограничить размер массива и упростить управление памятью.

Количество работников (col\_person) – тип int выбран для хранения количества работников на заводе. Это целое число позволяет легко отслеживать количество записей в массиве persons и выполнять операции, такие как добавление или удаление работников, а также проверку на переполнение массива.

Средний оклад (sr\_oklad) – тип float выбран для хранения среднего оклада работников завода. Использование типа с плавающей запятой позволяет точно представлять денежные значения, что важно для финансового анализа и отчетности. Это также позволяет выполнять арифметические операции для расчета среднего значения на основе окладов работников.

Файловая **база данных “Заводы”** разработана для выполнения основных операций с данными, таких как добавление и редактирование записей, поиск и сортировка, а также импорт и экспорт информации в файл. Эти возможности обеспечивают эффективное администрирование данных о рабочих и о заводе в целом.

Таблица 2.1 – описание основных функция программы

|  |  |
| --- | --- |
| Функции | Описание |
| Заполнение массива | add\_factory – добавляет новый завод в массив заводов, вызывая внутри себя add\_person для добавления персонала. |
| Вывод записей | print\_factory – выводит информацию о заводе и его работниках. print\_factories – выводит информацию всего файла массива структур. |
| Сохранение данных в файл | create\_factory\_file – сохраняет текущие данные массива в файл. |
| Очистка | clear\_factory\_file – очистка содержимого всего файла. clear\_factories – очистка текущих загруженных данных. |
| Загрузка данных из файла | read\_factory\_file – чтение информации из файла и заполнение массива factory. |
| Редактирование файла | edit\_factory – редактирование отдельных данных персонала заводов. |
| Поиск | search\_by\_quantity\_employees – поиск завода по, установленному пользователем, количеству рабочих. |
| Сортировка | compare\_number\_factory – сравнение номеров заводов для реализации сортировки. compare\_average\_salary - сравнение средних окладов заводов для реализации сортировки. |
| Меню | display\_menu() – вывод меню доступных операций. |

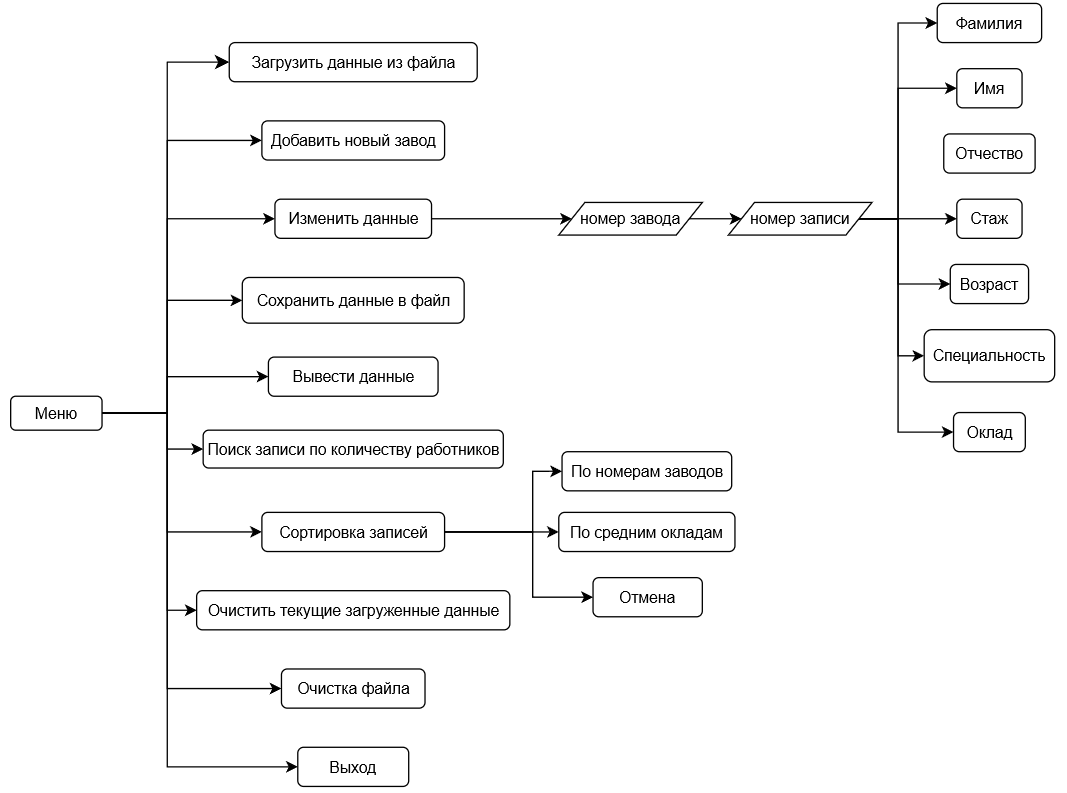
Интерфейс пользователя базы данных “Заводы” создан с акцентом на интуитивность и удобство использования. Его главная задача — обеспечить легкий доступ ко всем основным функциям программы, включая импорт данных из файла, добавление, редактирование, поиск, сортировку и сохранение информации.

При запуске программы на экране отображается меню с перечислением доступных действий, каждое из которых имеет свой номер для удобства выбора.

МЕНЮ

1. Загрузить данные из файла
2. Добавить новый завод
3. Изменить данные
   1. Номер завода
      1. Номер записи
         1. Фамилия
         2. Имя
         3. Отчество
         4. Возраст
         5. Специальность
         6. Оклад
         7. Стаж
4. Сохранить данные в файл
5. Вывести данные
6. Поиск записи по количеству работников
7. Сортировка записей
   1. По номерам заводов
   2. По среднему окладу завода
   3. Отмена
8. Очистить текущие загруженные данные
9. Очистка файла
10. Выход

Блок схема меню, представленная на рисунке 1.

  
Рисунок 2.1 – блок схема меню

* 1. **Описание функций**

Функция main является основным компонентом программы, управляющей базой данных "Заводы". Она отвечает за инициализацию, управление меню и выполнение различных операций с данными о заводах и их работниках.

В начале функции выводится заголовок, который информирует пользователя о названии курсовой работы, теме и авторе. Это помогает установить контекст и цель программы.

Функция выделяет память для массива, который будет хранить информацию о заводах. Если выделение памяти не удалось, программа выводит сообщение об ошибке и завершает выполнение. Также инициализируются переменные для хранения имени файла, выбора пользователя и количества заводов (char filename[50]; int choice, factory\_count = 0;)

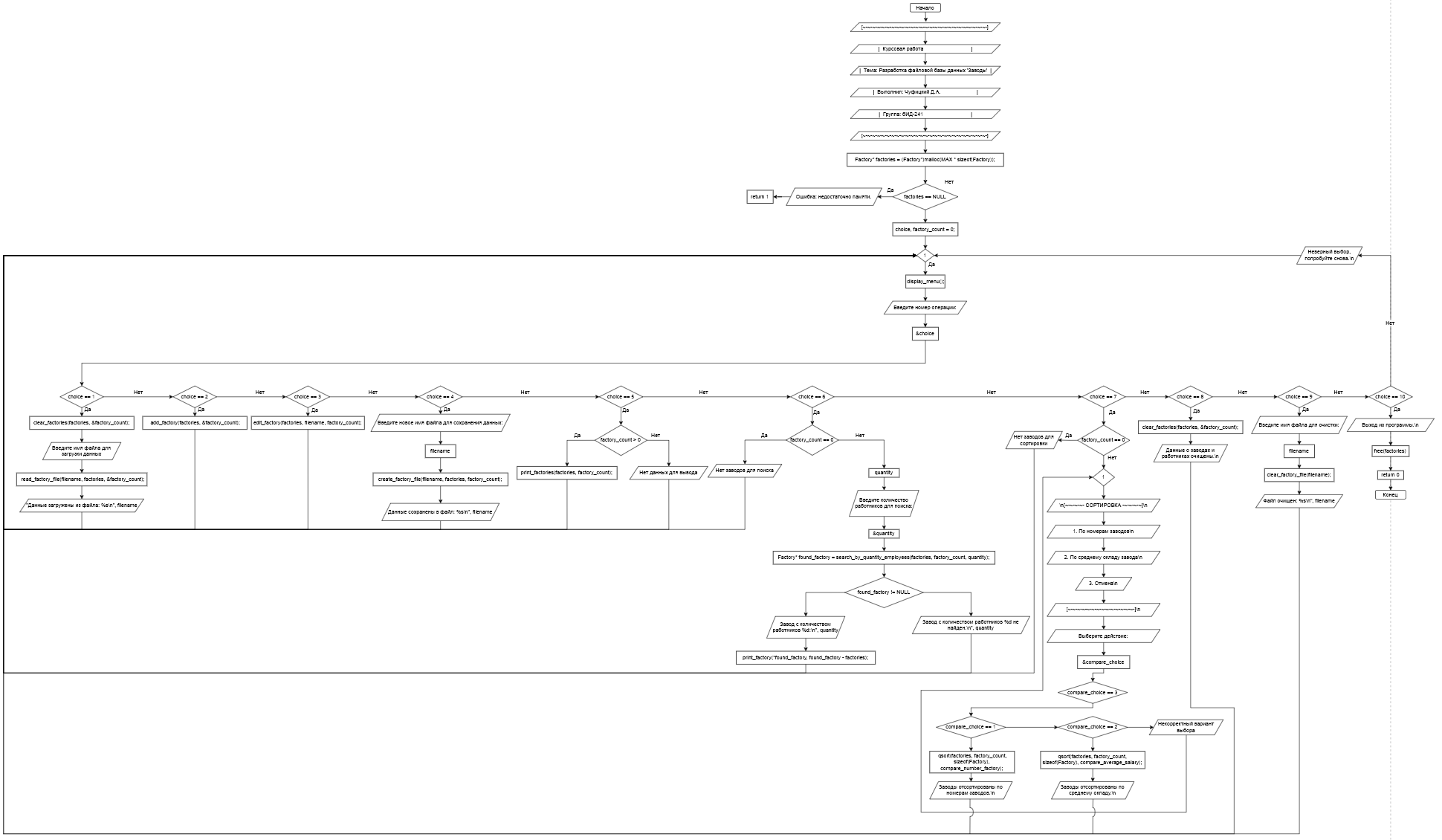
Главное меню: функция содержит бесконечный цикл, который обеспечивает отображение главного меню и выполнение операций до тех пор, пока пользователь не решит выйти из программы. В меню представлены различные операции, такие как загрузка данных, добавление нового завода, редактирование данных, сохранение в файл и вывод информации.

Обработка выбора пользователя: от выбора пользователя выполняются различные операции. Каждая операция обрабатывается с помощью конструкции switch, которая позволяет программе реагировать на ввод пользователя.

1. Загрузка данных из файла: программа очищает текущие данные и запрашивает имя файла для загрузки. После успешной загрузки выводится сообщение о завершении операции.
2. Добавление нового завода: пользователь может ввести данные о новом заводе, которые затем добавляются в массив.
3. Редактирование данных: программа предоставляет возможность изменять информацию о заводах и их работниках.
4. Сохранение данных в файл: пользователь может указать имя файла для сохранения текущих данных.
5. Вывод данных: программа выводит информацию о всех заводах, если они есть.
6. Поиск по количеству работников: пользователь может ввести количество работников для поиска соответствующего завода.
7. Сортировка записей: когда пользователь выбирает опцию сортировки, программа выводит меню с тремя вариантами: сортировка по номерам заводов, сортировка по среднему окладу и возможность отмены операции. После выбора программа выполняет сортировку и выводит обновлённый список заводов. Если пользователь выбирает “Отмена”, программа возвращается в главное меню.
8. Очистка данных: программа позволяет очистить текущие загруженные данные о заводах и работниках.
9. Очистка файла: пользователь может указать файл, который необходимо очистить.
10. Выход из программы: программа завершает выполнение, освобождая выделенную память

Завершение работы: если пользователь вводит неверный номер операции, программа уведомляет об этом и предлагает попробовать снова. Это обеспечивает удобство использования и предотвращает возможные сбои в работе программы.

Подробная блок схема функции main, представленная на рисунке 2.2.

  
Рисунок 2.2 – Подробная блок схема функции main

Функция **“add\_person”** предназначена для ввода данных о работнике и их сохранения в структуре person.

Принцип работы:

1. Запрашивает у пользователя информацию о человеке:
   1. Фамилия – fam
   2. Имя – im
   3. Отчество – ot
   4. Возраст – sr\_vsrt
   5. Специальность – speciality
   6. Оклад – zp
   7. Опыт(стаж) – opit
2. Заполняет соответствующие поля в структуре нового рабочего

Параметры:

struct person\* person – указатель на структуру person, который используется для передачи адреса конкретного работника в функции.

Функция **“add\_factory”** предназначена для добавления нового завода в массив factories.

Принцип работы:

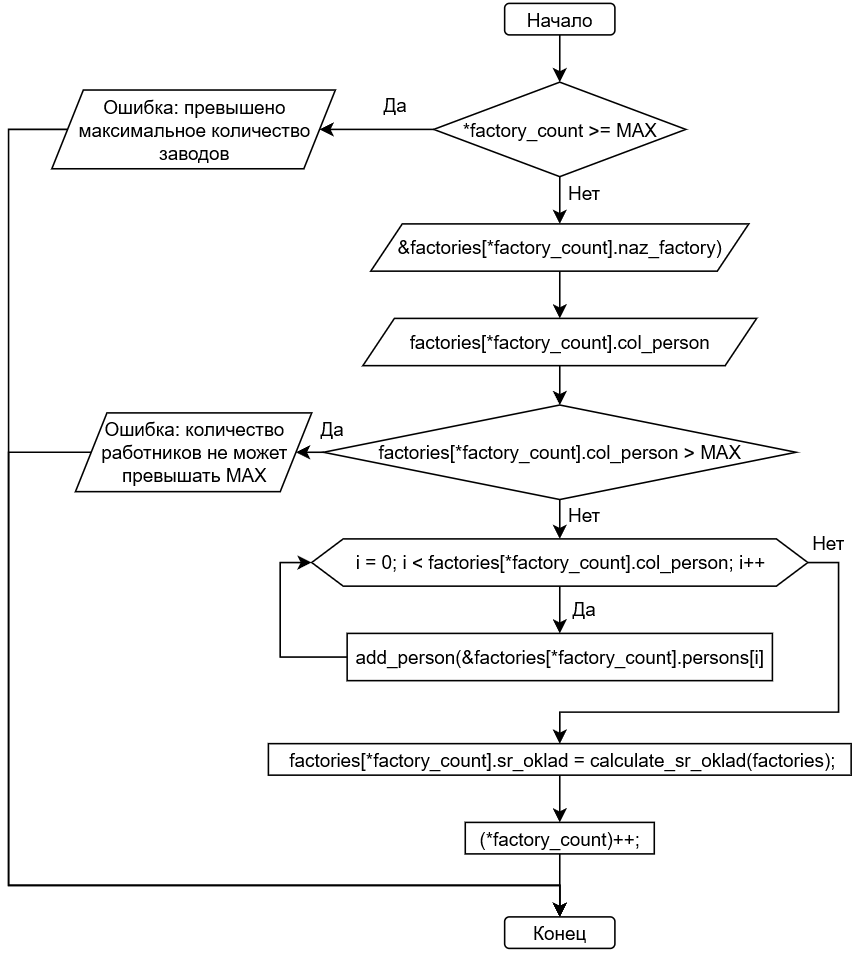
1. Проверяет, не превышает ли текущее количество заводов максимальное значение.  Если превышает, выводится сообщение об ошибке.
2. Запрашивает номер завода и количество работников, проверяя, чтобы оно не превышало максимальное значение.
3. После этого для каждого работника запрашиваются данные с помощью функции “add\_person”. В конце функция вычисляет средний оклад работников завода и увеличивает счётчик заводов.

Параметры:

Factory\* factories – указатель на массив структур Factory, который хранит информацию о заводах.

int\* factory\_count – указатель на целочисленную переменную, которая отслеживает текущее количество заводов в массиве.

Блок схема функции add\_factrory, представленная на рисунке 2.3.

  
Рисунок 2.3 – блок схема add\_factrory

Функция **“print\_factory”** предназначена для вывода информации о конкретном заводе и о его работниках.

Принцип работы:

1. Заголовок: начинает с вывода разделительной линии для визуального отделения данных.
2. Номер записи: выводит номер завода, используя индекс, увеличенный на единицу.
3. Данные о заводе: показывает номер завода, средний оклад и количество работников.
4. Данные о работниках: в цикле выводит информацию о каждом работнике, включая фамилию, имя, отчество, возраст, специальность, оклад и стаж.
5. Разделение: после каждого работника выводится разделительная линия для улучшения читаемости.

Параметры:

Factory factory — это структура, содержащая информацию о конкретном заводе, включая его номер, средний оклад и данные о работниках.

int index — это целое число, представляющее индекс завода в массиве. Он используется для отображения номера записи завода

Функция **“print\_factories”** предназначена для вывода информации о всех заводах массива.

Принцип работы :

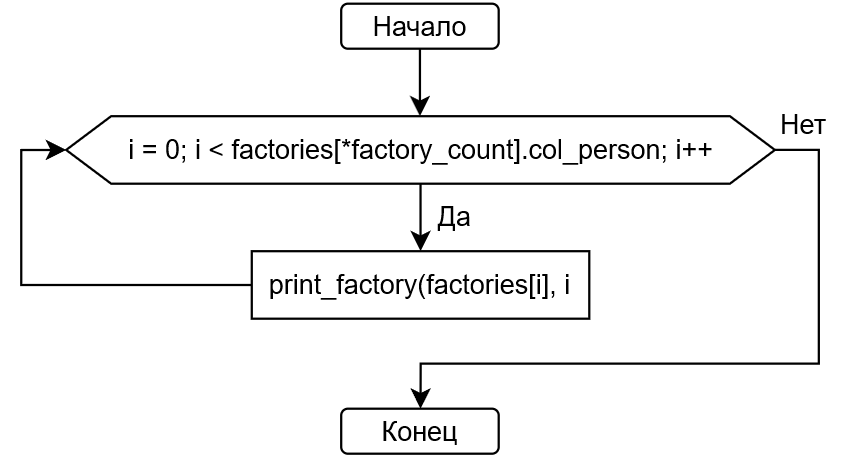
1. Цикл: Функция использует цикл for, чтобы пройтись по всем заводам массива.
2. Внутри цикла для каждого завода вызывается функция print\_factory, передавая текущий завод и его индекс. Это позволяет выводить информацию о каждом заводе по отдельности.

Параметры:

Factory\* factories — указатель на массив структур Factory, который содержит данные о заводах. Это позволяет функции работать с несколькими заводами.

int factory\_count — целое число, представляющее количество заводов в массиве. Этот параметр определяет, сколько заводов нужно обработать.

Блок схема функции print\_factories, представленная на рисунке 2.4.

  
Рисунок 2.4 – блок схема print\_factories

Функция **“create\_factory\_file”** организует процесс сохранения информации о заводах и их работниках в указанный файл.

Принцип работы:

1. Открытие файла: функция пытается открыть файл для записи с помощью fopen. Если файл не удаётся создать, выводится сообщение об ошибке, и функция завершает выполнение.
2. Проверка данных: если количество заводов (factory\_count) меньше 1, программа выводит сообщение о том, что нет данных для сохранения, закрывает файл и завершает выполнение.
3. Запись данных: В цикле for функция проходит по всем заводам и записывает их данные в файл: сначала записывается номер завода и количество работников. Затем для каждого работника записываются его фамилия, имя, отчество, возраст, специальность, оклад и стаж.
4. Закрытие файла: после завершения записи данных файл закрывается с помощью fclose.

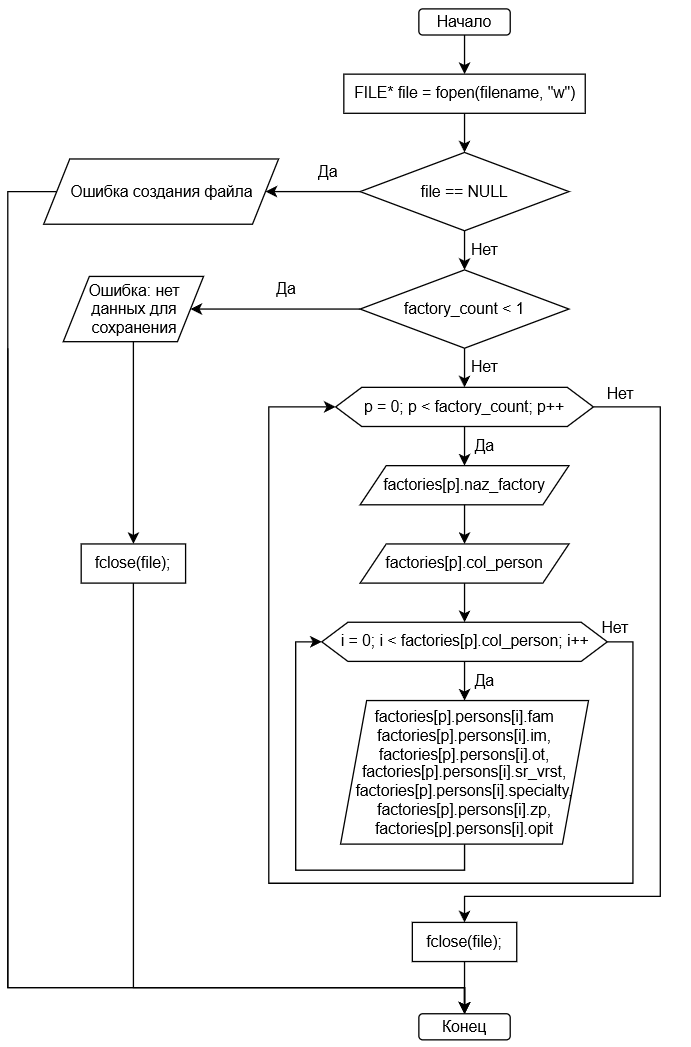
Параметры:

const char\* filename — строка, содержащая имя файла, в который будут записаны данные. Этот параметр передаётся как константный указатель, чтобы предотвратить изменения строки.

Factory\* factories — указатель на массив структур Factory, который содержит информацию о заводах. Это позволяет функции работать с несколькими заводами.

int factory\_count — целое число, представляющее количество заводов в массиве. Этот параметр определяет, сколько заводов нужно сохранить в файл.

Блок схема функции create\_factory\_file, представленная на рисунке 2.5.

  
Рисунок 2.5 – блок схема create\_factory\_file

Функция **“clear\_factory\_file”** эффективно очищает содержимое указанного файла, подготавливая его для дальнейшего использования.

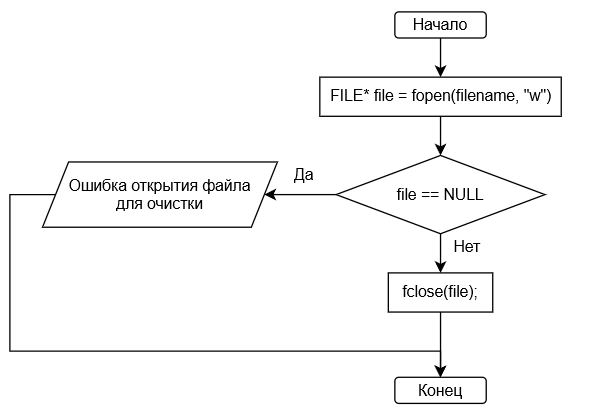
Принцип работы:

1. Открытие файла: функция пытается открыть файл для записи с помощью fopen с режимом "w". Этот режим автоматически очищает содержимое файла, если он существует. Если файл не удаётся открыть, выводится сообщение об ошибке, и функция завершает выполнение.
2. Закрытие файла: после успешного открытия файла (даже если он пустой) функция закрывает его с помощью fclose.

Параметры:

const char\* filename — строка, содержащая имя файла, который необходимо очистить. Этот параметр передаётся как константный указатель, чтобы предотвратить изменения строки.

Блок схема функции clear\_factory\_file, представленная на рисунке 2.6.

  
Рисунок 2.6 – блок схема clear\_factory\_file

Функция **“read\_factory\_file”** организует процесс загрузки информации о заводах и их работниках из файла, обеспечивая корректное считывание и обработку данных.

Принцип работы:

1. Открытие файла: функция пытается открыть файл для чтения с помощью fopen. Если файл не удаётся открыть, выводится сообщение об ошибке, и функция завершает выполнение.
2. Чтение данных: в бесконечном цикле функция считывает данные о заводах: сначала считывается номер завода (naz\_factory). Если чтение не удалось, цикл прерывается. Затем считывается количество работников (col\_person). Если чтение не удалось, выводится сообщение об ошибке, и функция завершает выполнение. Если количество работников превышает максимальное значение (MAX), выводится предупреждение, и количество устанавливается в MAX.
3. Чтение данных о работниках: внутри второго цикла функция считывает данные о каждом работнике, включая фамилию, имя, отчество, возраст, специальность, оклад и стаж. Если чтение данных о работнике не удалось, выводится сообщение об ошибке, и цикл прерывается.
4. Вычисление среднего оклада: после успешного чтения данных о работниках функция вызывает calculate\_sr\_oklad, чтобы вычислить средний оклад для завода.
5. Увеличение счётчика заводов: увеличивается значение factory\_count, чтобы отразить добавление нового завода.
6. Закрытие файла: после завершения чтения данных файл закрывается с помощью fclose.

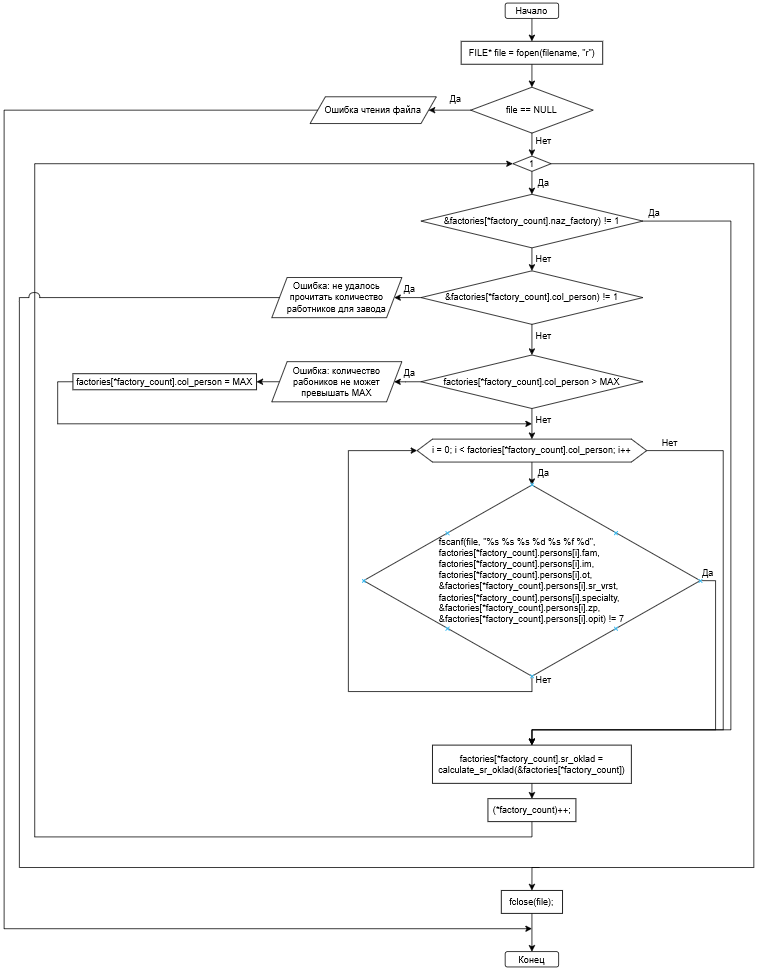
Параметры:

const char\* filename — строка, содержащая имя файла, из которого будут считываться данные. Этот параметр передаётся как константный указатель, чтобы предотвратить изменения строки.

Factory\* factories — указатель на массив структур Factory, в который будут загружены данные о заводах. Это позволяет функции работать с несколькими заводами.

int\* factory\_count — указатель на целочисленную переменную, которая отслеживает текущее количество заводов. Функция будет увеличивать это значение при добавлении новых заводов.

Блок схема функции read\_factory\_file, представленная на рисунке 2.7.

  
Рисунок 2.7 – блок схема read\_factory\_file

Функция **“edit\_factory”** организует процесс редактирования данных о работниках завода, обеспечивая удобный интерфейс для пользователя и сохранение изменений.

Принцип работы:

1. Проверка наличия данных: если количество заводов равно нулю, функция выводит сообщение о том, что нет данных для изменения, и завершает выполнение.
2. Вывод списка заводов: функция выводит список всех заводов с их номерами, чтобы пользователь мог выбрать, какой завод редактировать.
3. Выбор завода: пользователь вводит номер завода для изменения. Если введённый номер неверен, выводится сообщение об ошибке, и функция завершает выполнение.
4. Вывод работников завода: функция выводит список работников выбранного завода, чтобы пользователь мог выбрать, данные какого работника редактировать.
5. Выбор работника: пользователь вводит номер записи работника для изменения. Если номер неверен, выводится сообщение об ошибке, и функция завершает выполнение.
6. Выбор поля для редактирования: функция предлагает пользователю выбрать, какое поле работника он хочет изменить (фамилия, имя, отчество, возраст, специальность, оклад или стаж). Если введённый номер неверен, выводится сообщение об ошибке.
7. Редактирование данных: в зависимости от выбора пользователя, функция запрашивает новое значение для соответствующего поля и обновляет данные работника.
8. Пересчёт среднего оклада: после изменения данных функция вызывает calculate\_sr\_oklad, чтобы пересчитать средний оклад для завода.
9. Сохранение изменений: функция вызывает create\_factory\_file, чтобы сохранить обновлённые данные о заводах в указанный файл.
10. Сообщение об успешном сохранении: в конце выводится сообщение о том, что изменения успешно сохранены в файл.

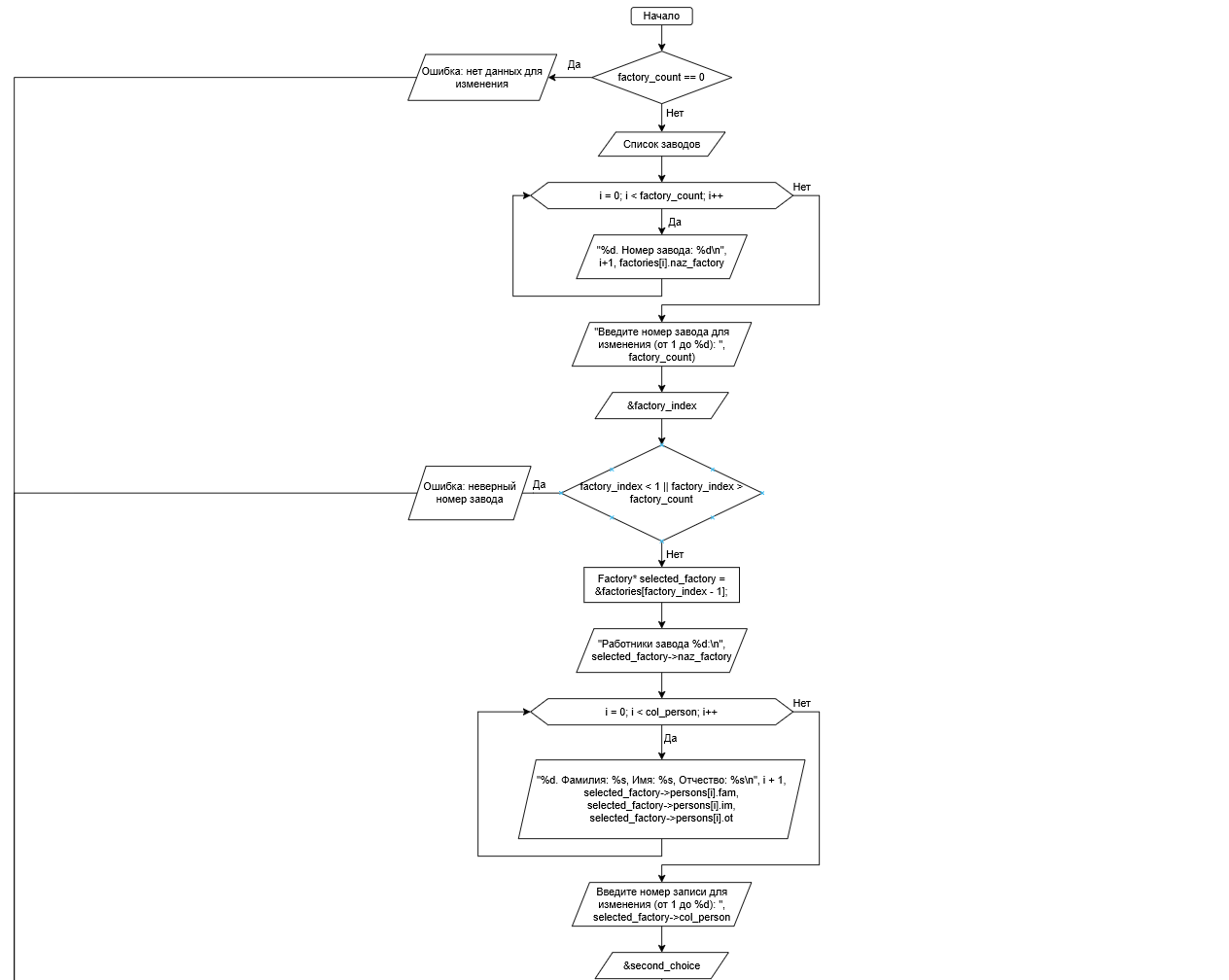
Параметры:

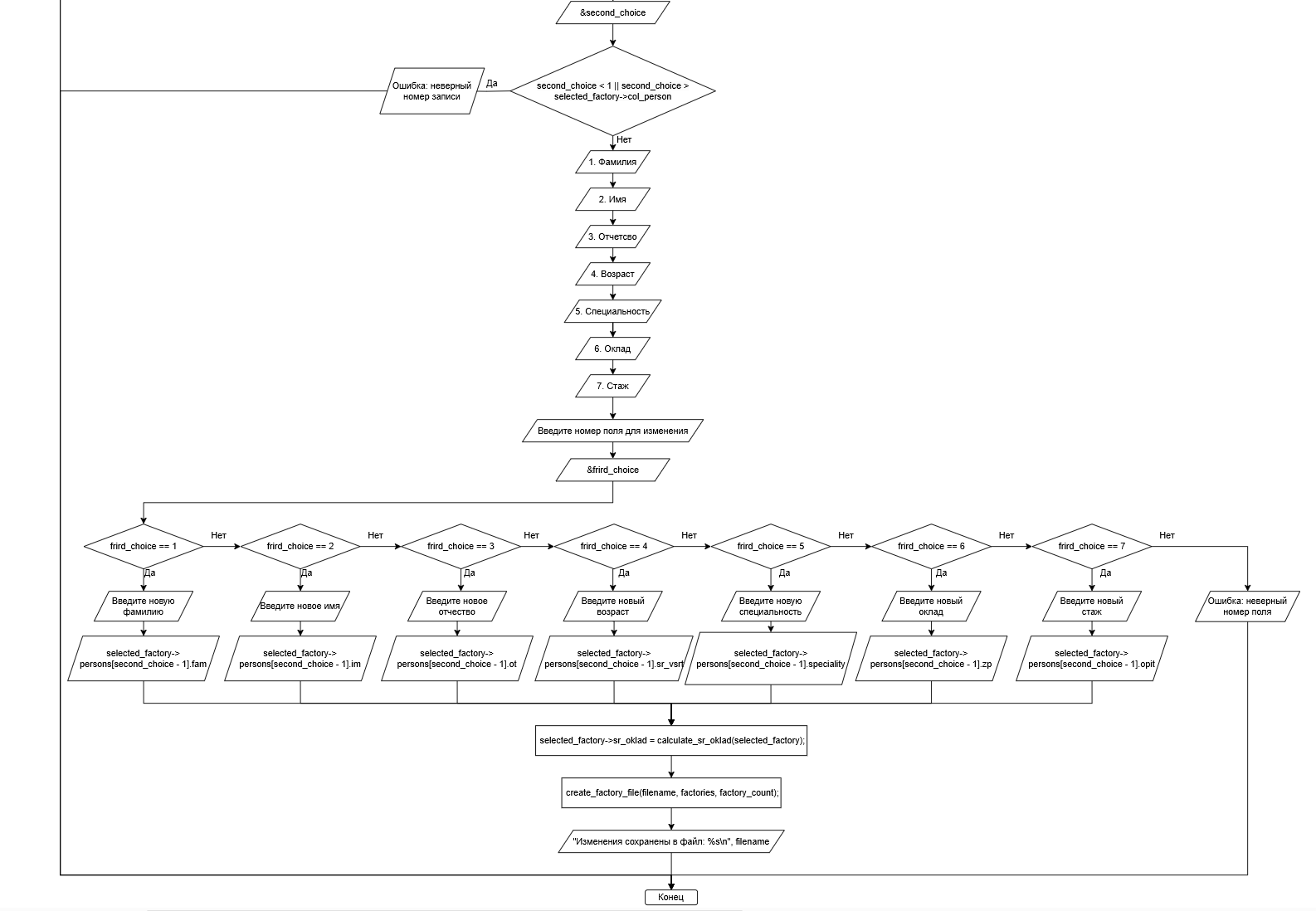
Factory\* factories — указатель на массив структур Factory, содержащий данные о заводах. Это позволяет функции работать с несколькими заводами.

const char\* filename — строка, содержащая имя файла, в который будут сохранены изменения. Этот параметр передаётся как константный указатель, чтобы предотвратить изменения строки.

int factory\_count — целое число, представляющее количество заводов в массиве. Этот параметр используется для проверки наличия данных и для управления выбором пользователя.

Блок схема функции edit\_factory, разбитая на 2 части, представленная на рисунке 2.8 и 2.9.

Рисунок 2.8 – первая часть блок схемы edit\_factory

  
Рисунок 2.9 – вторая часть блок схемы edit\_factory

Функция **“calculate\_sr\_oklad”** вычисляет средний оклад работников завода, обеспечивая корректный расчёт на основе имеющихся данных.

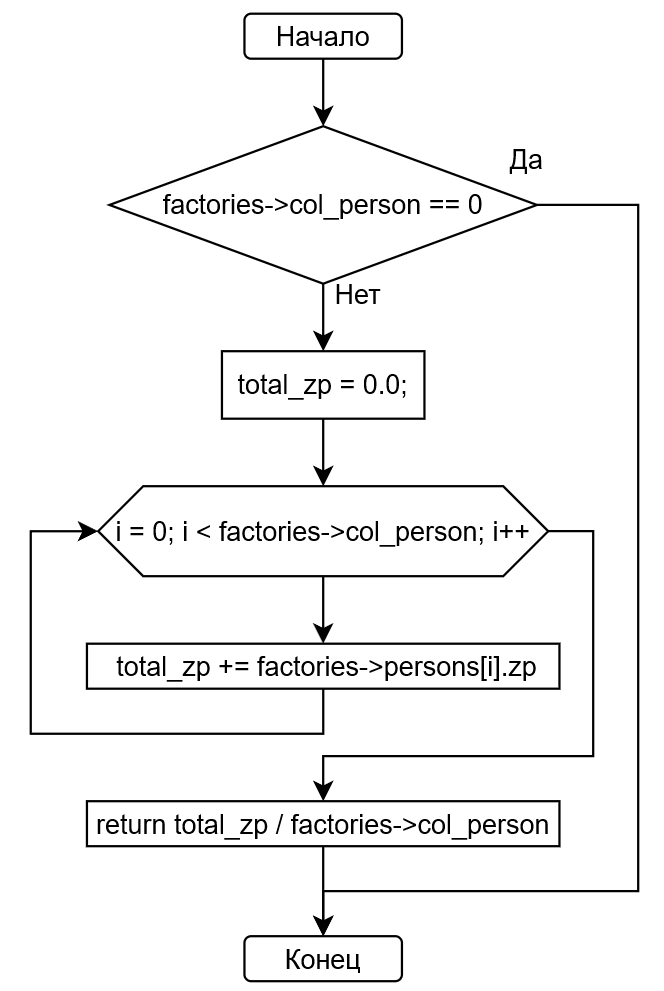
Принцип работы:

1. Проверка количества работников: если количество работников (col\_person) равно нулю, функция возвращает 0.0, так как делить на ноль невозможно.
2. Инициализация переменной: функция инициализирует переменную total\_zp для хранения общей суммы окладов работников.
3. Суммирование окладов: в цикле for функция проходит по всем работникам завода и суммирует их оклады (zp), добавляя каждое значение к total\_zp.
4. Вычисление среднего оклада: после завершения цикла функция делит общую сумму окладов на количество работников и возвращает результат.

Параметры:

Factory\* factories — указатель на структуру Factory, содержащую данные о заводе, включая информацию о работниках. Это позволяет функции получить доступ к данным о работниках для расчёта среднего оклада.

Блок схема функции calculate\_sr\_oklad, представленная на рисунке 2.10.

  
Рисунок 2.10 – блок схема calculate\_sr\_oklad

Функция **“clear\_factories”** предназначена для очистки данных о заводах, устанавливая количество заводов в ноль.

Принцип работы:

1. Сброс счётчика: функция устанавливает значение, на которое указывает factory\_count, в 0, что очищает информацию о заводах, так как количество заводов теперь равно нулю.

Параметры:

Factory\* factories — указатель на массив структур Factory, содержащий данные о заводах. Этот параметр не используется в данной функции, так как функция просто сбрасывает счётчик.

int\* factory\_count — указатель на целочисленную переменную, которая отслеживает текущее количество заводов. Функция изменяет это значение, устанавливая его в ноль.

Функция **“search\_by\_quantity\_employees”** предоставляет простой способ поиска завода по количеству работников, возвращая указатель на соответствующий завод или NULL, если такой завод не найден.

Принцип работы:

1. Цикл поиска: функция использует цикл for, чтобы пройтись по всем заводам.
2. Сравнение количества работников: внутри цикла функция проверяет, совпадает ли количество работников (col\_person) текущего завода с искомым значением. Если совпадение найдено, функция возвращает указатель на найденный завод.
3. Возврат NULL: если ни один завод не соответствует заданному количеству работников, функция возвращает NULL, указывая на то, что завод с таким количеством работников не найден.

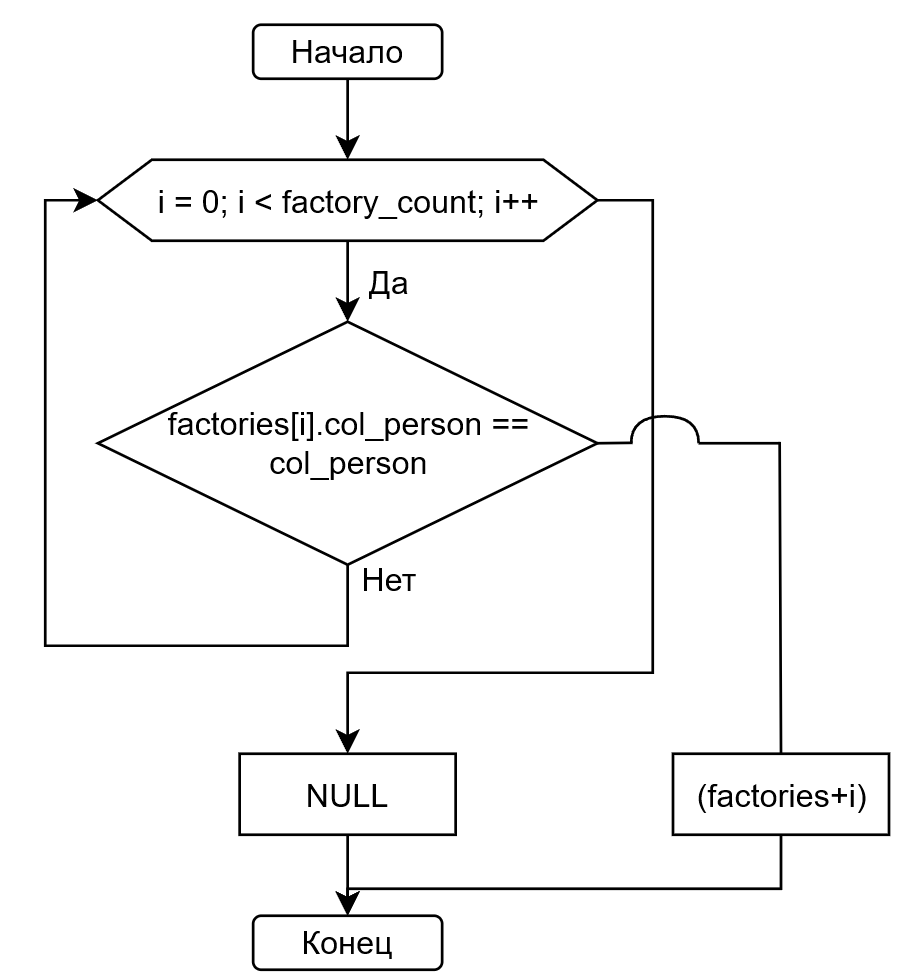
Параметры:

Factory\* factories — указатель на массив структур Factory, содержащий данные о заводах. Это позволяет функции получить доступ к информации о всех заводах.

int factory\_count — целое число, представляющее количество заводов в массиве. Этот параметр используется для определения границ цикла поиска.

int col\_person — целое число, представляющее количество работников, по которому осуществляется поиск. Функция ищет заводы с указанным количеством работников.

Блок схема функции search\_by\_quantity\_employees, представленная на рисунке 2.11.

  
Рисунок 2.11 – блок схема search\_by\_quantity\_employees

Функция **“compare\_number\_factory ”** используется для упорядочивания или же сравнения заводов по их номерам, возвращая значения, которые указывают на порядок между двумя объектами.

Принцип работы:

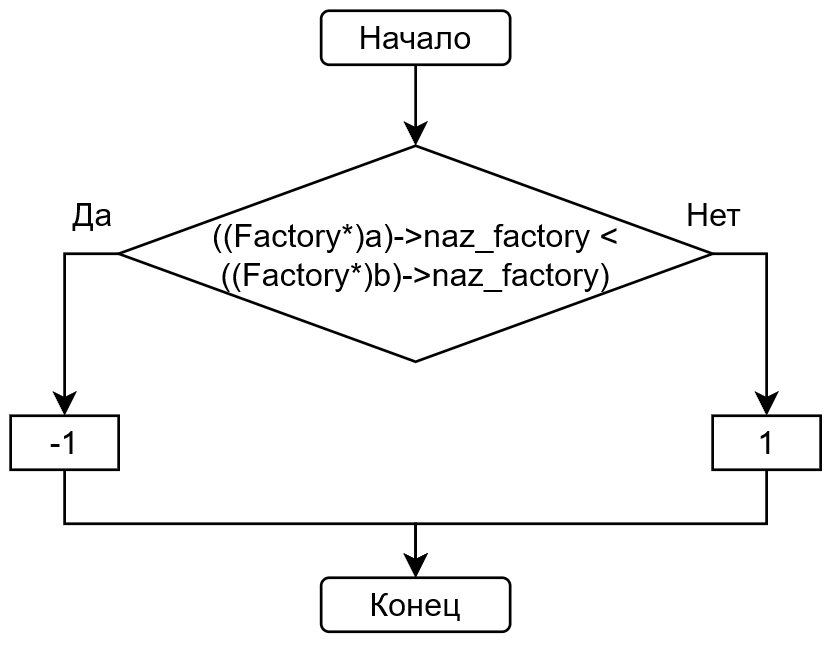
1. Сравнение номеров заводов: функция сравнивает номера заводов (naz\_factory) двух объектов. Если номер первого завода меньше номера второго, функция возвращает -1, иначе 1.

Параметры:

const void\* a — указатель на первый объект для сравнения, который будет приведён к типу Factory\*.

const void\* b — указатель на второй объект для сравнения, который также будет приведён к типу Factory\*

Блок схема функции compare\_number\_factory, представленная на рисунке 2.12.

  
Рисунок 2.12 – блок схема compare\_number\_factory

Функция **“compare\_average\_salary”** используется для упорядочивания или же сравнения заводов по средним окладам, возвращая значения, которые указывают на порядок между двумя объектами.

Принцип работы:

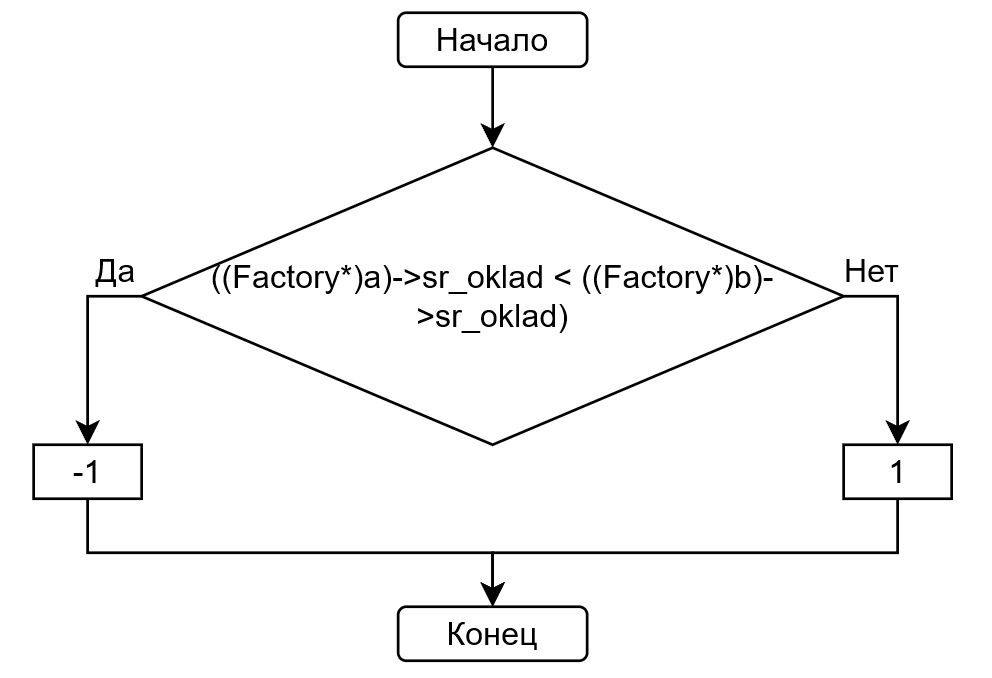
1. Сравнение средних окладов: функция сравнивает средние оклады (sr\_oklad) двух объектов. Если оклад первого завода меньше оклада второго, функция возвращает -1, иначе 1.

Параметры:

const void\* a — указатель на первый объект для сравнения, который будет приведён к типу Factory\*.

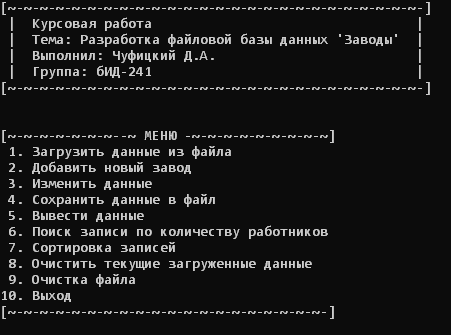
const void\* b — указатель на второй объект для сравнения, который также будет приведён к типу Factory\*

Блок схема функции compare\_average\_salary, представленная на рисунке 2.13.

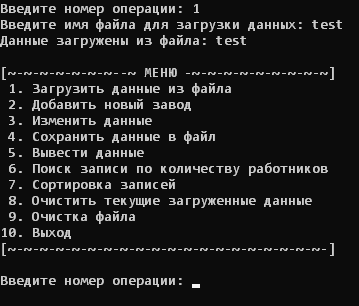
  
Рисунок 2.13 – блок схема compare\_average\_salary

1. **Тестирование программы**

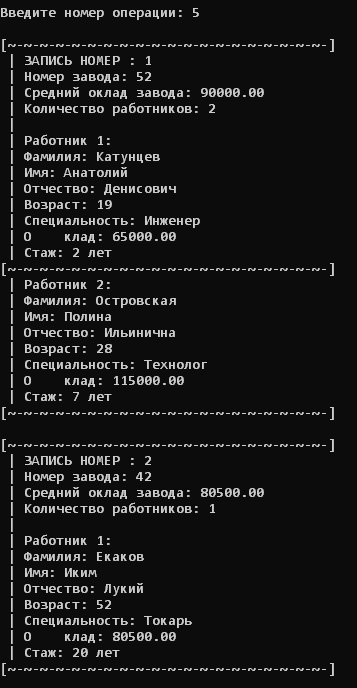
При запуске программы на экране отображается карточка с ФИО студента, его группой и названием программы, а также главное меню, содержащее ключевые функции приложения (рисунок 3.1).

  
Рисунок 3.1 – главное меню и карточка студента

Навигация по главному меню программы осуществляется путем ввода соответствующих цифр с клавиатуры в консоль для выбора пунктов меню. Начнём с 1 функции главного меню, для этого вводим в консоль “1” и нажимаем на “Enter”. Программа просит нас ввести имя файла, который будет загружен в текущий/действующий массив структур, например test. После чего, если все прошло успешно, соответствующее сообщение уведомит пользователя. После загрузки файла программа снова возвращает нас в главное меню (рисунок 3.2)

  
Рисунок 3.2 – тестирование первой функции

Так как до этого я вводил в файл test некоторые данные, я могу их вывести в консоли с помощью пятой функции главного меню. Ввожу в консоль “5”, нажимаю “Enter”. В консоли выводится все записи заводов, которые я добавлял в этот файл ранее (рисунок 3.3). Записи выводятся на экран в построчно в отдельных колонках, отделяясь от других записей соответствующей чертой.

  
Рисунок 3.3 – тестирование пятой функции

Далее, чтобы очистить наш текущий загруженный массив мы можем воспользоваться операцией номер “8”. Данная операция не удаляет наш файл. Если он нам понадобится, достаточно снова загрузить его с помощью первой функции. Вводим в консоль цифру “8”, программа говорит, что загруженные данные были очищены (рисунок 3.4).

  
Рисунок 3.4 – тестирование восьмой функции

Наши загруженные файлы были очищены, и теперь, чтобы удостовериться, при попытке снова вызвать функцию вывода “5” программа скажет, что у нас нет данных для вывода информации (рисунок 3.5).

  
Рисунок 3.5 – повторный вызов пятой функции

После того, как данные были очищены, можем воспользоваться опцией номер “2”, чтобы добавить завод. Вводим в консоль “2”, нажимаем “Enter”, сначала программа попросит ввести нас номер завода, вводим, например, 13, подтверждаем с помощью “Enter”, далее программа просит ввести количество персонала, который будет привязан к нашему заводу. Для примера вводим “2”. Теперь нам нужно поочередно ввести данные к каждому человеку, привязанному к заводу. Фамилию, имя, отчество, затем возраст, оклад и стаж. Сначала заполняются данные для первого работника, а затем такие же данные для второго (рисунок 3.6)

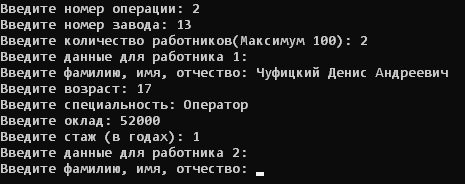
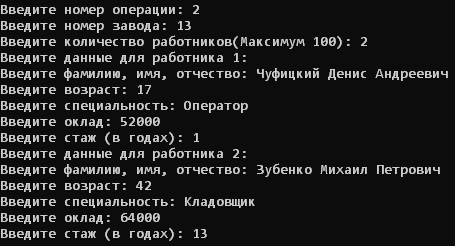


Рисунок 3.6 – заполнение завода

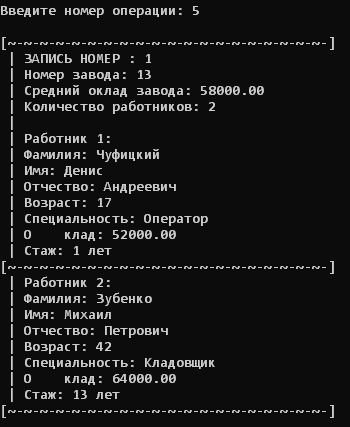
Продолжаем заполнять данные для второго работника (рисунок 3.7)

  
Рисунок 3.7 – полное заполнение завода

После заполнения данных нас переносит в главное меню, где мы можем вывести только что введенные данные, добавить еще одну запись завода или, например, сохранить эти данные в отдельный файл. Для того, чтобы сохранить данные, записываем в консоль цифру “4”. Программа просит нас придумать название для нашего файла, вводим, например, “record”. Выводится диагностическое сообщение о том, что данные были сохранены в файл “record” (рисунок 3.8).

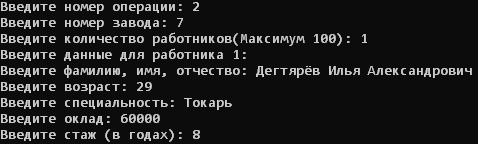
  
Рисунок 3.8 – тестирование четвёртой функции

Чтобы проверить, сохранились ли наши данные, вводим в консоль цифру “8”, чтобы очистить текущий/загруженный массив. Затем пишем в консоль цифру “1” для загрузки данных из файла. Вводим “record” и вызываем пятую функцию, для вывода только записанного файла. В итоге получаем результат, представленный на рисунке 3.9.

  
Рисунок 3.9 – проверка сохранения файла

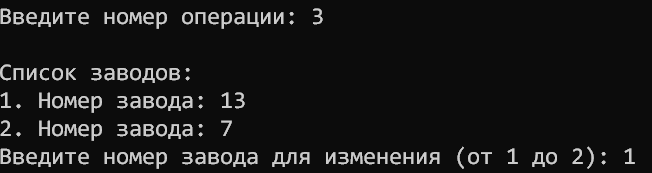
Далее, пожалуй, добавим еще один завод в наш массив, это понадобится для более подробного понятия функции редактирования, а также для поиска записи или сортировки записей.

Повторяем пункт с добавлением завода, но на этот раз добавим 1 человека в персонал завода (рисунок 3.10).

  
Рисунок 3.10 – добавление нового завода

Можем обновить файл, снова сохранив его с помощью функции “4”. Впишем такое же название файла (record).

Воспользуемся опцией номер “3”, изменением данных файла. Вводим в консоль “3”, у нас появляется список заводов, вводим в консоль один из них, например 1 (рисунок 3.11).

  
Рисунок 3.11 – список заводов для редактирования

Далее у нас выводится список работников завода номер 13, выбираем первого работника (рисунок 3.12).

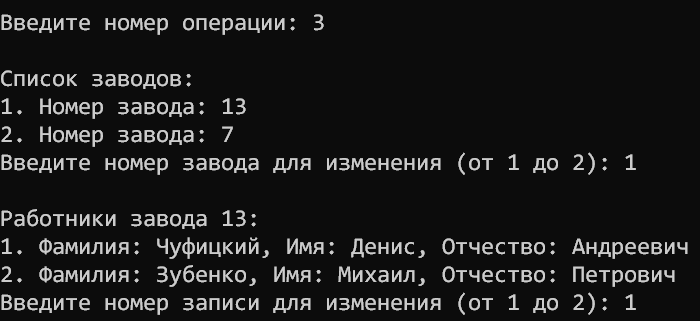
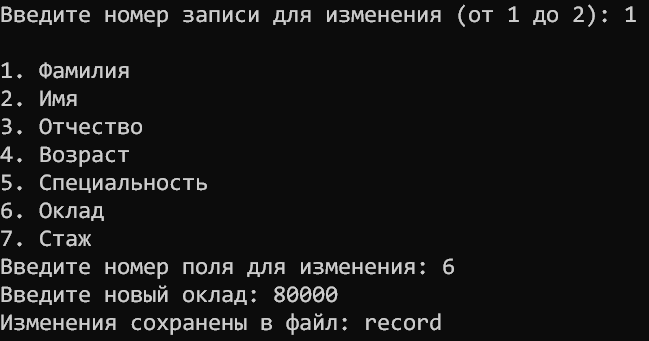
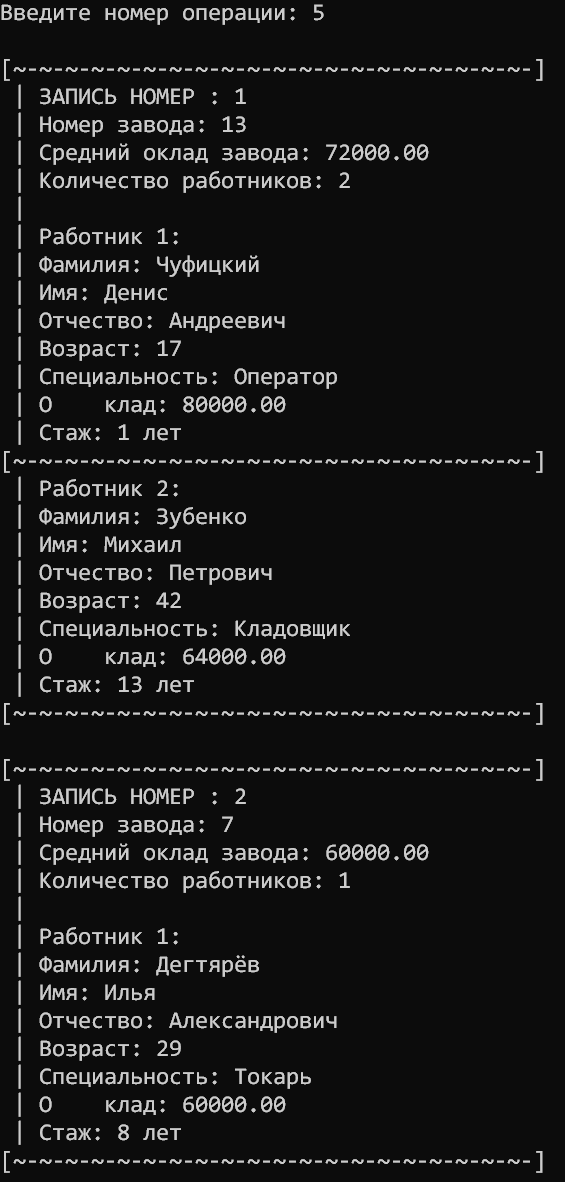


Рисунок 3.12 – работники завода номера 13

Затем выводится список всех характеристик работника, которые можно изменить. Вводим в консоль “6” и у программа просит нас ввести новый оклад работника. После ввода новой информации программа пишет, что изменения были сохранены в наш файл record (рисунок 3.13).

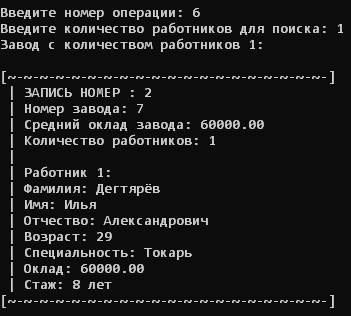
  
Рисунок 3.13 – изменение записи работника

Может снова проверить изменения, вызовом функции номер “5” (рисунок 3.14).

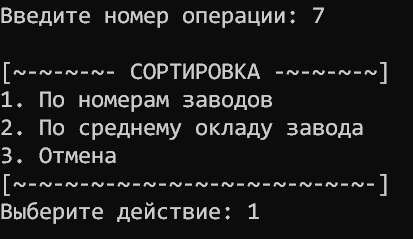
  
Рисунок 3.14 – отредактированный файл record

Как мы видим, оклад первого рабочего поменялся, а вместе с этим поменялся и средний оклад всего завода.

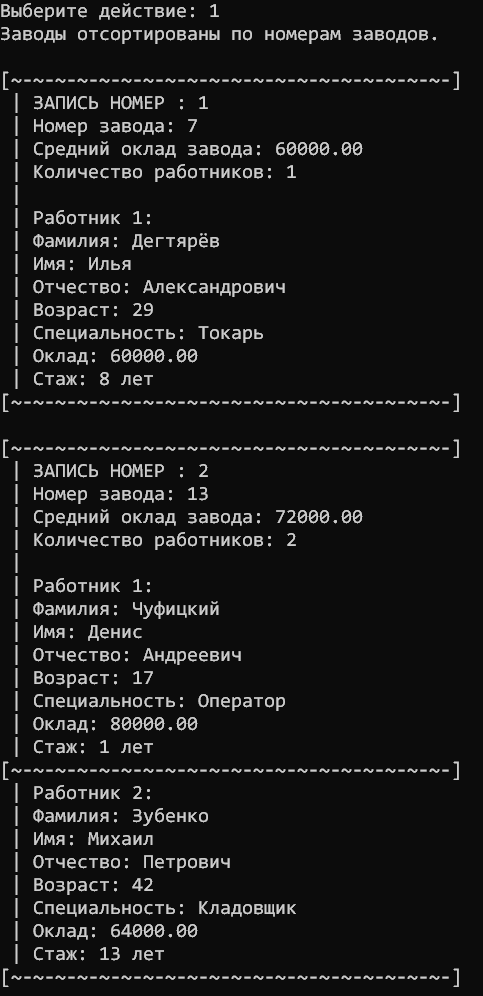
Вызовем функцию номер “6”, которая ищет завод в массиве заводов по количеству персонала, программа просит ввести количество персонала завода, вводим, например 1, после этого выводится, найденный программой, завод, с 1 работником. Этим заводом является вторая запись завода (рисунок 3.15).

  
Рисунок 3.15 – тестирование шестой функции

Воспользуемся функцией сортировки, для этого в консоль запишем цифру “7”. Программа выведет дополнительное меню, в котором можно выбрать сортировку по номерам или по средним окладам заводов (рисунок 3.17). Вводим, например, 1, сортировку по номерам заводов.

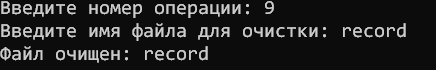
  
Рисунок 3.17 – выбор варианта сортировки\

Далее выводится обновлённый массив с заводами, отсортированными от меньшего к большему номерами заводов. В нашем случае 2 завода поменяются местами (рисунок 3.18).

  
Рисунок 3.18 – отсортированный массив заводов

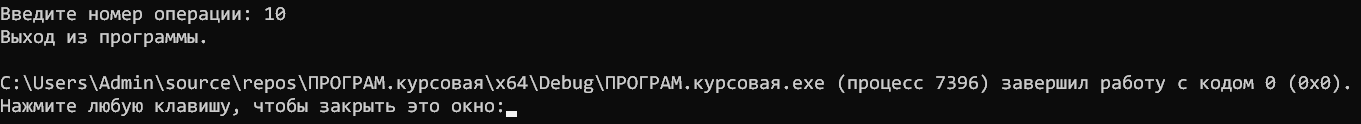
Чтобы вернутся в главное меню, вводим в консоль цифру “3”.

Все основные опции главного меню были продемонстрированы, теперь выбираем опцию полной очистки файла под номером “9”, вводим название файла “record”. Программа выводит сообщение о том, что файл был очищен (рисунок 3.19).

  
Рисунок 3.19 – полная очистка файла record

Чтобы убедиться, что файл был очищен, вводим в консоль “1” (загружаем данные из файла), записываем название файла “record”, затем выбираем опцию “5” (вывод заводов). Программа выводит сообщение о том, что у нее нет данных для вывода.

Остаётся последняя функция, функция выхода из программы. Для этого в консоль записываем число “10”, нажимаем “Enter” (рисунок 3.20).

  
Рисунок 3.20 – завершение программы.

**ЗАКЛЮЧЕНИЕ**

Созданная база данных "Заводы" соответствует требованиям для структурированного хранения, поиска, сортировки и редактирования информации о заводах. Она является компактным и эффективным инструментом для управления данными. Выполнение курсового проекта дало возможность улучшить навыки работы с файлами, указателями, структурами данных и взаимодействия с пользователем через консольный интерфейс. Данная программа может служить основой для дальнейшего расширения функциональности и разработки более сложных систем управления данными.

**СПИСОК ИСПОЛЬЗОВАНЫХ ИСТОЧНИКОВ**

1. Лекционные занятия “Основы программирования и алгоритмизации”.
2. Практические занятия “Основы программирования и алгоритмизации”.
3. Методические указания курсовой работы
4. "Язык программирования Си" — Брайан Керниган, Деннис Ритчи Классическая книга, написанная создателями языка Си, охватывающая основы и продвинутые темы.
5. "Программирование на языке Си" — Стивен Кохан Подробное руководство, которое предлагает четкие объяснения и множество примеров для практики.
6. "Си. Полное руководство" — Герберт Шилдт Обширное руководство по языку Си, охватывающее как основы, так и более сложные темы.
7. "Язык программирования Си для начинающих" — Игорь Кузнецов Книга, ориентированная на начинающих программистов, с простыми объяснениями и примерами.
8. "Си. Учебный курс" — А. В. Шевченко Учебное пособие, которое охватывает основные концепции языка Си и предлагает практические задания.
9. "Си. Учебник для программистов" — В. В. Петров Книга, которая подходит как для новичков, так и для опытных программистов, с акцентом на практические аспекты.
10. "Программирование на Си" — А. И. Костюков Краткое и доступное руководство по языку Си с примерами и задачами.
11. "Си. Справочник" — П. Дж. Плагер Справочник по стандартной библиотеке языка Си, включая примеры использования.
12. "Си для профессионалов" — Питер ван дер Линден Книга для более опытных программистов, рассматривающая тонкости и секреты языка Си.
13. Онлайн-ресурс: Хабр, платформа с множеством статей, уроков и примеров кода по языку Си и другим языкам программирования. <https://habr.com/ru/hub/c/>

**ПРИЛОЖЕНИЕ**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <string.h>

#define MAX 100

struct person {

char fam[20];

char im[20];

char ot[20];

int sr\_vrst;

char specialty[50];

float zp;

int opit; // Стаж (в годах)

};

typedef struct person Person;

struct factory {

int naz\_factory;

struct person persons[MAX];

int col\_person;

float sr\_oklad;

};

typedef struct factory Factory;

/\* Объявление функций\*/

void add\_factory(Factory\* factories, int\* factory\_count);

// Добавляет новый завод в массив заводов.

void add\_person(struct person\* person);

// Запрашивает и добавляет данные о новом работнике.

void print\_factory(Factory factory, int index);

// Выводит информацию о заводе и его работниках.

void print\_factories(Factory\* factories, int factory\_count);

// Выводит информацию о вс ех заводах.

void create\_factory\_file(const char\* filename, Factory\* factories, int factory\_count); // Сохраняет данные о заводах в файл.

void clear\_factory\_file(const char\* filename);

// Очищает содержимое указанного файла.

void read\_factory\_file(const char\* filename, Factory\* factories, int\* factory\_count); // Загружает данные о заводах из файла.

void edit\_factory(Factory\* factories, const char\* filename, int factory\_count); // Позволяет редактировать данные о заводе и его работниках.

float calculate\_sr\_oklad(Factory\* factories);

// Вычисляет средний оклад работников завода.

void clear\_factories(Factory\* factories, int\* factory\_count);

// Очистить текущие загруженные данные

Factory\* search\_by\_quantity\_employees(Factory\* factories, int factory\_count, char\* col\_person);

// Ищет завод по количеству работников.

int compare\_number\_factory(const void\* a, const void\* b);

// Сравнивает два завода по их номерам.

int compare\_average\_salary(const void\* a, const void\* b);

// Сравнивает два завода по среднему окладу.

void display\_menu();

// Выводит меню доступных операций.

int main() {

printf("[~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-]\n");

printf(" | Курсовая работа |\n");

printf(" | Тема: Разработка файловой базы данных 'Заводы' |\n");

printf(" | Выполнил: Чуфицкий Д.А. |\n");

printf(" | Группа: бИД-241 |\n");

printf("[~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-]\n\n");

Factory\* factories = (Factory\*)malloc(MAX \* sizeof(Factory));

if (factories == NULL) {

printf("Ошибка: недостаточно памяти.\n");

return 1;

}

char filename[50];

int choice, factory\_count = 0;

while (1) {

display\_menu();

printf("Введите номер операции: ");

scanf("%d", &choice);

switch (choice) {

case 1:

clear\_factories(factories, &factory\_count);

printf("Введите имя файла для загрузки данных: ");

scanf("%s", filename);

read\_factory\_file(filename, factories, &factory\_count);

printf("Данные загружены из файла: %s\n", filename);

break;

case 2:

add\_factory(factories, &factory\_count);

break;

case 3:

edit\_factory(factories, filename, factory\_count);

break;

case 4:

printf("Введите новое имя файла для сохранения данных: ");

scanf("%s", filename);

create\_factory\_file(filename, factories, factory\_count);

printf("Данные сохранены в файл: %s\n", filename);

break;

case 5:

if (factory\_count > 0) {

print\_factories(factories, factory\_count);

}

else {

printf("Нет данных для вывода.\n");

}

break;

case 6: {

if (factory\_count == 0) {

printf("Нет заводов для поиска.\n");

break;

}

int quantity;

printf("Введите количество работников для поиска: ");

scanf("%d", &quantity);

Factory\* found\_factory = search\_by\_quantity\_employees(factories, factory\_count, quantity);

if (found\_factory != NULL) {

printf("Завод с количеством работников %d:\n", quantity);

print\_factory(\*found\_factory, found\_factory - factories);

// Вычитание указателей в Си возвращает количество элементов между ними

}

else {

printf("Завод с количеством работников %d не найден.\n", quantity);

}

break;

}

case 7:

if (factory\_count == 0) {

printf("Нет заводов для сортировки.\n");

break;

}

int compare\_choice;

while (1) {

printf("\n[~-~-~-~- СОРТИРОВКА -~-~-~-~]\n");

printf("1. По номерам заводов\n");

printf("2. По среднему окладу завода\n");

printf("3. Отмена\n");

printf("[~-~-~-~-~-~-~-~-~-~-~-~-~-~-]\n");

printf("Выберите действие: ");

scanf("%d", &compare\_choice);

if (compare\_choice == 3) {

break;

}

switch (compare\_choice) {

case 1:

qsort(factories, factory\_count, sizeof(Factory), compare\_number\_factory);

printf("Заводы отсортированы по номерам заводов.\n");

break;

case 2:

qsort(factories, factory\_count, sizeof(Factory), compare\_average\_salary);

printf("Заводы отсортированы по среднему окладу.\n");

break;

default:

printf("Некорректный вариант выбора.\n");

continue;

}

print\_factories(factories, factory\_count);

}

break;

case 8:

clear\_factories(factories, &factory\_count);

printf("Данные о заводах и работниках очищены.\n");

break;

case 9:

printf("Введите имя файла для очистки: ");

scanf("%s", filename);

clear\_factory\_file(filename);

printf("Файл очищен: %s\n", filename);

break;

case 10:

printf("Выход из программы.\n");

free(factories); // Освобождение памяти

return 0;

default:

printf("Неверный выбор, попробуйте снова.\n");

break;

}

}

}

/\* Функции \*/

// Функция добавления завода в массив заводов.

void add\_factory(Factory\* factories, int\* factory\_count) {

if (\*factory\_count >= MAX) {

printf("Ошибка: превышено максимальное количество заводов.\n");

}

printf("Введите номер завода: ");

scanf("%d", &factories[\*factory\_count].naz\_factory);

printf("Введите количество работников(Максимум %d): ", MAX);

scanf("%d", &factories[\*factory\_count].col\_person);

if (factories[\*factory\_count].col\_person > MAX) {

printf("Ошибка: количество работников не может превышать %d.\n", MAX);

}

for (int i = 0; i < factories[\*factory\_count].col\_person; i++) {

printf("Введите данные для работника %d:\n", i + 1);

add\_person(&factories[\*factory\_count].persons[i]);

}

factories[\*factory\_count].sr\_oklad = calculate\_sr\_oklad(factories);

(\*factory\_count)++;

}

// Функция для добавления данных о новом работнике.

void add\_person(struct person\* person) {

printf("Введите фамилию, имя, отчество: ");

scanf("%s %s %s", person->fam, person->im, person->ot);

while (1) {

printf("Введите возраст: ");

scanf("%d", &person->sr\_vrst);

if (person->sr\_vrst < 0) {

printf("Ошибка: возраст не может быть отрицательным. Попробуйте снова.\n");

}

else {

break;

}

}

printf("Введите специальность: ");

scanf("%s", person->specialty);

while (1) {

printf("Введите оклад: ");

scanf("%f", &person->zp);

if (person->zp < 0) {

printf("Ошибка: оклад не может быть отрицательным. Попробуйте снова.\n");

}

else {

break;

}

}

while (1) {

printf("Введите стаж (в годах): ");

scanf("%d", &person->opit);

if (person->opit < 0) {

printf("Ошибка: стаж не может быть отрицательным. Попробуйте снова.\n");

}

else {

break;

}

}

}

// Функция для вывода информации о заводе и его работниках.

void print\_factory(Factory factory, int index) {

printf("\n[~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-]\n");

printf(" | ЗАПИСЬ НОМЕР : %d", index + 1);

printf("\n | Номер завода: %d\n", factory.naz\_factory);

printf(" | Средний оклад завода: %.2f\n", factory.sr\_oklad);

printf(" | Количество работников: %d\n |\n", factory.col\_person);

for (int i = 0; i < factory.col\_person; i++) {

printf(" | Работник %d:\n", i + 1);

printf(" | Фамилия: %s\n", factory.persons[i].fam);

printf(" | Имя: %s\n", factory.persons[i].im);

printf(" | Отчество: %s\n", factory.persons[i].ot);

printf(" | Возраст: %d\n", factory.persons[i].sr\_vrst);

printf(" | Специальность: %s\n", factory.persons[i].specialty);

printf(" | Оклад: %.2f\n", factory.persons[i].zp);

printf(" | Стаж: %d лет\n", factory.persons[i].opit);

printf("[~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-]\n");

}

}

// Функция для вывода информации о всех заводах.

void print\_factories(Factory\* factories, int factory\_count) {

for (int i = 0; i < factory\_count; i++) {

print\_factory(factories[i], i);

}

}

// Функция для сохранения данных о заводах в файл.

void create\_factory\_file(const char\* filename, Factory\* factories, int factory\_count) {

FILE\* file = fopen(filename, "w");

if (file == NULL) {

perror("Ошибка создания файла");

return;

}

if (factory\_count < 1) {

printf("Ошибка: нет данных для сохранения.\n");

fclose(file);

return;

}

for (int p = 0; p < factory\_count; p++) {

fprintf(file, "%d\n", factories[p].naz\_factory);

fprintf(file, "%d\n", factories[p].col\_person);

for (int i = 0; i < factories[p].col\_person; i++) {

fprintf(file, "%s %s %s %d %s %.2f %d\n",

factories[p].persons[i].fam,

factories[p].persons[i].im,

factories[p].persons[i].ot,

factories[p].persons[i].sr\_vrst,

factories[p].persons[i].specialty,

factories[p].persons[i].zp,

factories[p].persons[i].opit);

}

}

fclose(file);

}

// Функция очистки содержимого в указанном файле.

void clear\_factory\_file(const char\* filename) {

FILE\* file = fopen(filename, "w");

if (file == NULL) {

perror("Ошибка открытия файла для очистки");

return;

}

fclose(file);

}

// Функция чтения данных из файла.

void read\_factory\_file(const char\* filename, Factory\* factories, int\* factory\_count) {

FILE\* file = fopen(filename, "r");

if (file == NULL) {

perror("Ошибка чтения файла");

return;

}

while (1) {

if (fscanf(file, "%d", &factories[\*factory\_count].naz\_factory) != 1) {

break;

}

if (fscanf(file, "%d", &factories[\*factory\_count].col\_person) != 1) {

printf("Ошибка: не удалось прочитать количество работников для завода %d\n", factories[\*factory\_count].naz\_factory);

fclose(file);

return;

}

if (factories[\*factory\_count].col\_person > MAX) {

printf("Ошибка: количество работников не может превышать %d. Установлено в %d.\n", MAX, MAX);

factories[\*factory\_count].col\_person = MAX;

}

for (int i = 0; i < factories[\*factory\_count].col\_person; i++) {

if (fscanf(file, "%s %s %s %d %s %f %d",

factories[\*factory\_count].persons[i].fam,

factories[\*factory\_count].persons[i].im,

factories[\*factory\_count].persons[i].ot,

&factories[\*factory\_count].persons[i].sr\_vrst,

factories[\*factory\_count].persons[i].specialty,

&factories[\*factory\_count].persons[i].zp,

&factories[\*factory\_count].persons[i].opit) != 7) {

break;

}

}

factories[\*factory\_count].sr\_oklad = calculate\_sr\_oklad(&factories[\*factory\_count]);

(\*factory\_count)++;

}

fclose(file);

}

// Функция для редактирования данных о заводе и его работниках.

void edit\_factory(Factory\* factories, const char\* filename, int factory\_count) {

if (factory\_count == 0) {

printf("Нет данных для изменения.\n");

return;

}

printf("\nСписок заводов:\n");

for (int i = 0; i < factory\_count; i++) {

printf("%d. Номер завода: %d\n", i + 1, factories[i].naz\_factory);

}

int factory\_index;

printf("Введите номер завода для изменения (от 1 до %d): ", factory\_count);

scanf("%d", &factory\_index);

if (factory\_index < 1 || factory\_index > factory\_count) {

printf("Ошибка: неверный номер завода.\n");

return;

}

Factory\* selected\_factory = &factories[factory\_index - 1];

printf("\nРаботники завода %d:\n", selected\_factory->naz\_factory);

for (int i = 0; i < selected\_factory->col\_person; i++) {

printf("%d. Фамилия: %s, Имя: %s, Отчество: %s\n", i + 1,

selected\_factory->persons[i].fam,

selected\_factory->persons[i].im,

selected\_factory->persons[i].ot);

}

int second\_choice;

printf("Введите номер записи для изменения (от 1 до %d): ", selected\_factory->col\_person);

scanf("%d", &second\_choice);

if (second\_choice < 1 || second\_choice > selected\_factory->col\_person) {

printf("Ошибка: неверный номер записи.\n");

return;

}

int frird\_choice;

printf("\n1. Фамилия\n");

printf("2. Имя\n");

printf("3. Отчество\n");

printf("4. Возраст\n");

printf("5. Специальность\n");

printf("6. Оклад\n");

printf("7. Стаж\n");

printf("Введите номер поля для изменения: ");

scanf("%d", &frird\_choice);

switch (frird\_choice) {

case 1:

printf("Введите новую фамилию: ");

scanf("%s", selected\_factory->persons[second\_choice - 1].fam);

break;

case 2:

printf("Введите новое имя: ");

scanf("%s", selected\_factory->persons[second\_choice - 1].im);

break;

case 3:

printf("Введите новое отчество: ");

scanf("%s", selected\_factory->persons[second\_choice - 1].ot);

break;

case 4:

printf("Введите новый возраст: ");

scanf("%d", &selected\_factory->persons[second\_choice - 1].sr\_vrst);

break;

case 5:

printf("Введите новую специальность: ");

scanf("%s", selected\_factory->persons[second\_choice - 1].specialty);

break;

case 6:

printf("Введите новый оклад: ");

scanf("%f", &selected\_factory->persons[second\_choice - 1].zp);

break;

case 7:

printf("Введите новый стаж: ");

scanf("%d", &selected\_factory->persons[second\_choice - 1].opit);

break;

default:

printf("Ошибка: неверный номер поля.\n");

return;

}

selected\_factory->sr\_oklad = calculate\_sr\_oklad(selected\_factory);

create\_factory\_file(filename, factories, factory\_count);

printf("Изменения сохранены в файл: %s\n", filename);

}

// Функция для вычисления среднего оклада работников завода.

float calculate\_sr\_oklad(Factory\* factories) {

if (factories->col\_person == 0) {

return 0.0;

}

float total\_zp = 0.0;

for (int i = 0; i < factories->col\_person; i++) {

total\_zp += factories->persons[i].zp;

}

return total\_zp / factories->col\_person;

}

// Функция для очистки исходных данных о заводах.

void clear\_factories(Factory\* factories, int\* factory\_count) {

\*factory\_count = 0;

}

// Функция поиска завода по количеству персонала.

Factory\* search\_by\_quantity\_employees(Factory\* factories, int factory\_count, int col\_person) {

for (int i = 0; i < factory\_count; i++) {

if (factories[i].col\_person == col\_person)

return (factories + i);

}

return NULL;

}

// Функция сортировки по номеру завода.

int compare\_number\_factory(const void\* a, const void\* b) {

if (((Factory\*)a)->naz\_factory < ((Factory\*)b)->naz\_factory) {

return -1;

}

else

return 1;

}

int compare\_average\_salary(const void\* a, const void\* b) {

if (((Factory\*)a)->sr\_oklad < ((Factory\*)b)->sr\_oklad) {

return -1;

}

else

return 1;

}

// Функция для вывода меню доступных операций.

void display\_menu() {

printf("\n[~-~-~-~-~-~-~--~ МЕНЮ -~-~-~-~-~-~-~-~-~]\n");

printf(" 1. Загрузить данные из файла\n");

printf(" 2. Добавить новый завод\n");

printf(" 3. Изменить данные\n");

printf(" 4. Сохранить данные в файл\n");

printf(" 5. Вывести данные\n");

printf(" 6. Поиск записи по количеству работников\n");

printf(" 7. Сортировка записей\n");

printf(" 8. Очистить текущие загруженные данные\n");

printf(" 9. Очистка файла\n");

printf("10. Выход\n");

printf("[~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-~-]\n\n");

}