

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий и управления в технических системах
(полное название института)

кафедра «Информационные системы»
(полное название кафедры)

Пояснительная записка

к курсовой работе

на тему Manufacture. Программа учета и мониторинга списка
сотрудников предприятия.

по дисциплине Алгоритмизация и программирование

Выполнил: студент II курса, группы: ИС/6-22-о

Направления подготовки (специальности) 09.03.02

Информационные системы и технологии
(код и наименование направления подготовки (специальности))

профиль (специализация) _____

Клышко Никиты Александровича
(фамилия, имя, отчество студента)

Дата допуска к защите « _____ » _____ 20 18 г.

Руководитель _____

(подпись)

(инициалы, фамилия)

20 18 г.

Аннотация

В данной пояснительной записке содержится проектная документация приложения Manufactury – программы учета и мониторинга списка сотрудников предприятия, разработанного в рамках курсового проекта по дисциплине «Алгоритмизация и программирование».

Во введении обозначена тема курсового проекта, проблема, которая должна быть решена в результате разработки, определены цели и задачи.

В основной части документа описано назначение и область применения программы, приведены технические характеристики программы, в том числе используемые математические методы, метод организации входных, выходных и промежуточных данных, структурные схемы алгоритмов программы и описание к ним, состав технических и программных средств, описан процесс использования разработанной программы.

В заключении делается вывод о результатах проделанной работы и дается оценка разработанного приложения.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ	5
ВВЕДЕНИЕ.....	6
1 НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ.....	8
2 ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ ПРОГРАММЫ	9
2.1 Постановка задачи	9
2.2 Применяемые математические методы	9
2.3 Описание и обоснование метода организации входных, выходных и промежуточных данных	11
2.4 Архитектура программы	12
2.5 Описание алгоритмов программы.....	12
2.5.1 Реализация полиморфизма на основе структур языка С	12
2.5.2 Реализация табличного редактора программы	14
2.5.3 Реализация функции сортировки записей	15
2.6 Обоснование состава технических и программных средств	18
2.6.1 Выбор интегрированной среды разработки	18
2.6.2 Выбор TUI библиотеки.....	18
2.6.3 Выбор библиотеки коллекций (динамических структур).....	20
3 ИСПОЛЬЗОВАНИЕ ПРОГРАММЫ	21
3.1 Системные требования программы.....	21
3.2 Загрузка и запуск программы	21
3.3 Тестирование программы.....	22
3.3.1 Раздел меню «Файл».....	22
3.3.2 Раздел меню «Правка»	25
3.3.3 Раздел меню «Инструменты»	28

3.3.4 Функции табличного редактора	30
ВЫВОДЫ.....	31
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.....	32
Приложение А. Исходный код программы	33
Приложение Б. Алгоритмы функций	52

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

- CUA – Common User Access, типовой пользовательский интерфейс
- MVP – Model-View-Presenter (рус. Модель-Вид-Представитель)
- TUI – Text-based user interface (рус. Текстовый пользовательский интерфейс)
- ИС – информационная система
- ОС – операционная система
- ПО – программное обеспечение
- ЯП – язык программирования

ВВЕДЕНИЕ

Темой курсового проекта является разработка положения Manufactury – программы учета и мониторинга списка сотрудников предприятия. Техническое задание на реализацию программы утверждено Севастопольским государственным университетом 07.09.18 г.

Данный курсовой проект направлен на решение проблемы учета и мониторинга списка сотрудников предприятия. Этот проект достаточно актуален, так как большинство современных программ подобного класса являются коммерческими и не имеют бесплатного тарифа. Начинающие предприниматели часто имеют значительно ограниченные ресурсы и не могут позволить себе приобретать дорогостоящее программное обеспечение (далее – ПО) для управления предприятием. Временные ресурсы так же ограничены, что вынуждает прибегать к использованию различных информационных систем (далее – ИС), которые помогают автоматизировать некоторые процессы предприятия, в том числе процессы учета сотрудников.

Целью данного курсового проекта является разработка приложения Manufactury, позволяющего автоматизировать учет сотрудников предприятия, имеющего функцию генерации отчета о количестве сотрудников с группировкой по нескольким параметрам.

Для достижения поставленной цели были определены следующие задачи:

1. Разработать оптимальную модель данных, позволяющую хранить, изменять, обрабатывать и отображать необходимую информацию.
2. Выделить в приложении основные составляющие на основе архитектурного шаблона проектирования Model-View-Presenter (далее – MVP).
3. Разработать библиотеку визуальных компонентов для реализации текстового пользовательского интерфейса (далее – TUI).

4. Реализовать компоненты программы на языке программирования (далее – ЯП) С (Си), используя выбранные вспомогательные библиотеки (см. 2.6).
5. Протестировать и отладить разработанное приложение.

1 НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

Назначение приложения Manufactury – создание, изменение, сортировка списка сотрудников предприятия, генерация отчета о количестве сотрудников с группировкой по разрядам и стажу работы.

Приложение может быть использовано на предприятиях с различным количеством сотрудников. Часто бывает так, что крупные предприятия, имеющие большое количество сотрудников, представляют собой сложную иерархическую структуру. В таких предприятиях использование данного приложения несколько ограничено, так как реализация сложной иерархии сотрудников не предусмотрена в нем. Этот недостаток может быть компенсирован использованием приложения для определенного уровня иерархической структуры.

Таким образом, приложение Manufactury имеет достаточно широкую область применения в сфере управления предприятием.

2 ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ ПРОГРАММЫ

2.1 Постановка задачи

Структура записей входного файла имеет следующий вид: табельный номер, Ф.И.О. (30 символов), год рождения пол (булев тип), профессия (10 символов), стаж работы, разряд рабочего, номер цеха, номер участка, сумма заработной платы. Вид выходной таблицы представлен таблицей 2.1.

Таблица 2.1 – Вид выходной таблицы

Стаж работы	Количество рабочих по разрядам			Всего
	1	2	3	
До 6				
С 6 до 11				
С 11 до 16				
С 16 до 21				
С 21 до 25				
Свыше 25				

2.2 Применяемые математические методы

При реализации визуального компонента панели пролистывания (ScrollBar) были выведены формулы для вычисления координат и размера ползунка панели на основе общего количества данных и номера текущей страницы. ScrollBar имеет определенную высоту (*height*), но это число включает в себя также высоту кнопок управления (scroll arrows), поэтому в вычислениях используется значение *height* – 2.

При изменении общего количества данных (*count*) необходимо вычислить достаточное количество позиций ползунка (*size*) для охвата всего набора данных. Для этого нужно посчитать количество страниц (*pages*), на которых можно разместить все данные. Так как может получиться неполная последняя страница, то округлим результат деления вверх до ближайшего целого.

$$pages = \left\lceil \frac{count}{height} \right\rceil$$

Теперь, зная количество страниц (*pages*) можно вычислить значение *size*. Если страниц больше, чем максимальное количество позиций ползунка (*height* – 2), то это значит, что на одно смещение ползунка будет приходиться пролистывание нескольких страниц, таким образом *size* должно быть максимальным (*height* – 2). Если количество данных равно 0, то и *pages* = 0, следовательно, вся панель будет заполнена ползунком, который не может перемещаться, то есть *size* = 1. В остальных случаях *size* = *pages*.

$$size = \begin{cases} height - 2, & pages \geq height - 2 \\ 1, & pages = 0 \\ pages & \end{cases}$$

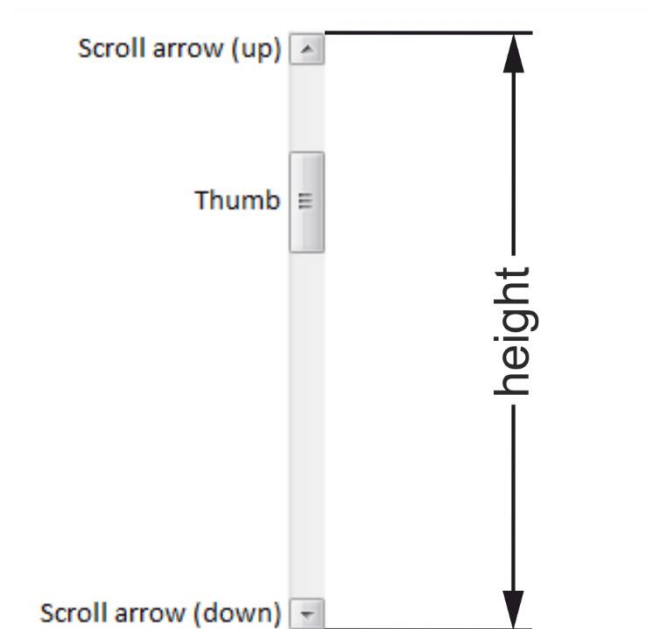


Рисунок 2.1 – Схематическое изображение панели пролистывания

После того, как получено значение *size*, необходимо вычислить размер ползунка (*thumb*) по формуле:

$$thumb = (height - 2) - size + 1$$

Когда размер ползунка известен, необходимо вычислить коэффициент прокрутки, то есть на какое количество позиций (*step*) в массиве данных приходится одна позиция ползунка:

$$step = \left\lceil \frac{count}{size} \right\rceil$$

При изменении позиции (*number*) в массиве данных необходимо вычислить новую позицию (*pos*) ползунка:

$$pos = \left\lfloor \frac{number}{step} \right\rfloor$$

2.3 Описание и обоснование метода организации входных, выходных и промежуточных данных

В программе все данные хранятся в динамических структурах типа `ArrayList` (динамический массив, тип `Array` в библиотеке `CollectionsC`). Эта структура поддерживает тот же набор операций, что и обычный список, но обеспечивает доступ к элементам по индексу за константное время, как у обычного массива. Это позволяет производить вставку в отсортированную структуру за время $O(\log n)$ (см. 2.5.3).

Структура одной записи представлена на рисунке 2.2.

Employee
+ int id
+ wchar_t surname[]
+ wchar_t name[]
+ wchar_t patronymic[]
+ short yob
+ bool gender
+ wchar_t profession[]
+ char experience
+ ProfClass class
+ char department
+ char plot
+ int salary

Рисунок 2.2 – Структура одной записи

Исходное строковое поле ФИО было решено разделить на 3 отдельных поля: фамилия, имя, отчество. Это позволило сохранить атомарность данных и упростить их обработку.

2.4 Архитектура программы

При проектировании приложения, за основу был взят архитектурный шаблон проектирования MVP, который подразумевает разделение программы на данные(Model) и Вид(View), связь между которыми реализуется посредством Представителя (Presenter) (рис. 2.3).

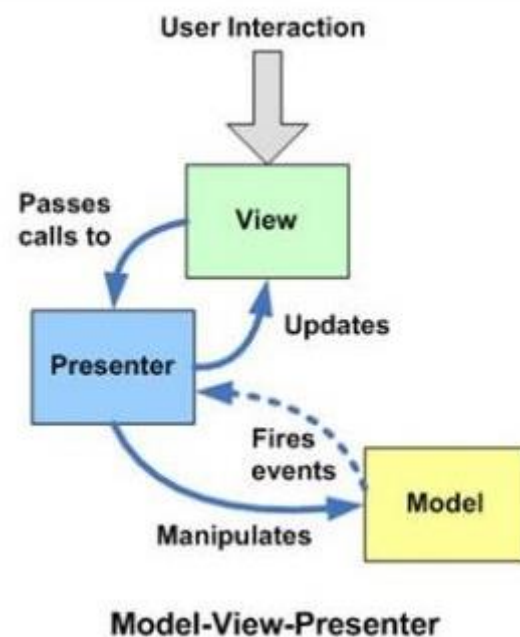


Рисунок 2.3 – Шаблон проектирования MVP

Причем логика обработки данных находится в Представителе, а Вид отвечает только за отображение данных, реализуя определенный интерфейс (контракт).

Таким образом, можно изменять визуальное представление программы независимо от логики программы.

2.5 Описание алгоритмов программы

2.5.1 Реализация полиморфизма на основе структур языка C

Реализация полноценного пользовательского интерфейса предусматривает наличие различных управляющих элементов пользовательского интерфейса, например, Label, Edit, Button и др.

Каждый управляющий элемент может иметь собственные функции поведения (т.н. методы), например, отрисовка(Show), скрытие(Hide), обработка

нажатия клавиши(OnClick) и т.п. Для обобщения алгоритмов библиотеки компонентов было необходимо использовать механизм полиморфизма. Так как язык C, в отличие от C++, не является объектно-ориентированным ЯП, то не имеет встроенной поддержки полиморфизма на основе классов. Поэтому необходимо было реализовать механизм полиморфизма на основе структур ЯП C (рис. 2.4).

Component
+ ... + void* spec + void (* Show)(Component* component) + void (* Hide)(Component* component) + void (* OnKeyClick)(Component* handle, int key, unsigned long modifiers) + bool (* OnFocusGet)(Component* handle) + bool (* OnFocusChange)(Component* handle) + void (* OnFocusLost)(Component* handle)

Рисунок 2.4 – Базовая структура Component

За основу реализации полиморфизма был взят мощный инструмент ЯП C – указатели на функции. Таким образом, за счет указателей на функции поведения визуального компонента, был обеспечен полиморфизм методов визуального компонента. Причем, эти функции принимают в качестве одного из аргументов указатель на структуру самого компонента, что позволяет получить доступ к полям конкретного экземпляра компонента в теле функции поведения.

Но каждый компонент также должен иметь возможность хранить дополнительные данные или даже добавлять собственные методы. Для этого в структуру Component было добавлено поле указателя на неопределенный тип (void) (рис. 2.4). Это позволило помещать в это поле указатель на собственный тип визуального компонента, который может содержать все необходимые дополнительные поля.

2.5.2 Реализация табличного редактора программы

Главное окно программы представляет собой табличный редактор записей. Каждая ячейка этой таблицы представляет собой, в зависимости от типа отображаемых/редактируемых данных, компонент Edit, Select или Button.

Компонент Button используется для столбца «Табельный номер», так как это ключевое поле и его изменение приводит к изменению не только визуального представления данных, но и порядка данных в структуре памяти. Поэтому, для упрощения отслеживания необходимости обновления структуры данных в памяти, редактирование значения в столбце «Табельный номер» вынесено в отдельное окно редактирования записи.

Компонент Select используется для столбцов с ограниченным набором значений, то есть для столбцов «Пол» и «Разряд».

Компонент Edit используется для всех остальных столбцов, причем для числовых столбцов («Год рождения», «Опыт работы» и т.д.) ограничивается ввод нецифровых символов при помощи механизма фильтрации ввода этого компонента. Алгоритм применения фильтрующей функции изображен на рисунке 2.5.

Для удобного использования такой сложной визуальной структуры были реализованы методы отображения данных страницы записей из произвольного массива. За счет этого была обеспечена простота реализации функции сортировки таблицы по любым полям структуры записи (см. 2.5.3), так как для отображения сортировки необходимо только передать новый отсортированный массив в функцию отображения данных.

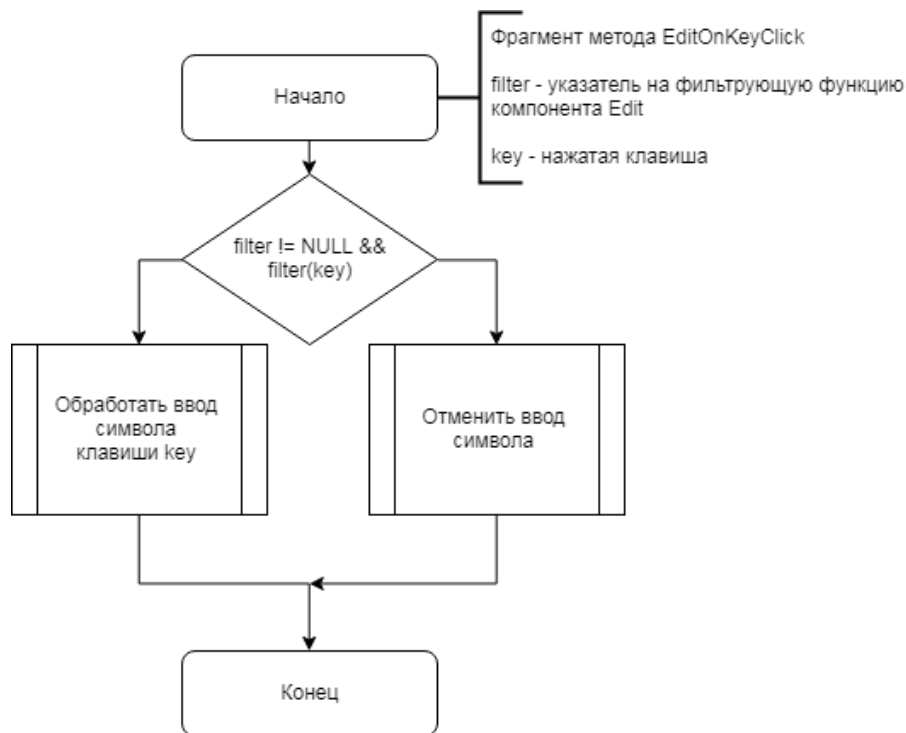


Рисунок 2.5 – Алгоритм механизма фильтрации ввода компонента Edit

2.5.3 Реализация функции сортировки записей

Для добавления в программу функции сортировки данных был разработан компонент ColumnLabel (Заголовок столбца), который дает возможность пользователю выбирать столбец, по которому происходит сортировка, а также направление сортировки.

Как показано на рисунке 2.6, ресурсоемкая операция сортировки данных происходит в Представителе, а отображение уже отсортированной информации происходит с помощью обратного вызова функции отображения данных на Виде (см. 2.5.2).

Так как в режиме сортировки программа не запрещает редактирование данных, то нужно было разработать алгоритм изменения данных записи таким образом, чтобы не нарушалась сортировка и измененная запись оставалась видимой пользователю. Для этого было реализовано расширения для библиотеки CollectionsC, позволяющее добавить запись в массив, не нарушая порядок сортировки.

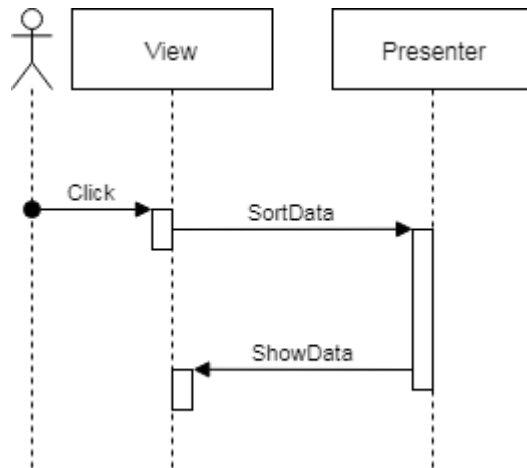


Рисунок 2.6 – Диаграмма последовательности вызовов при использовании функции сортировки

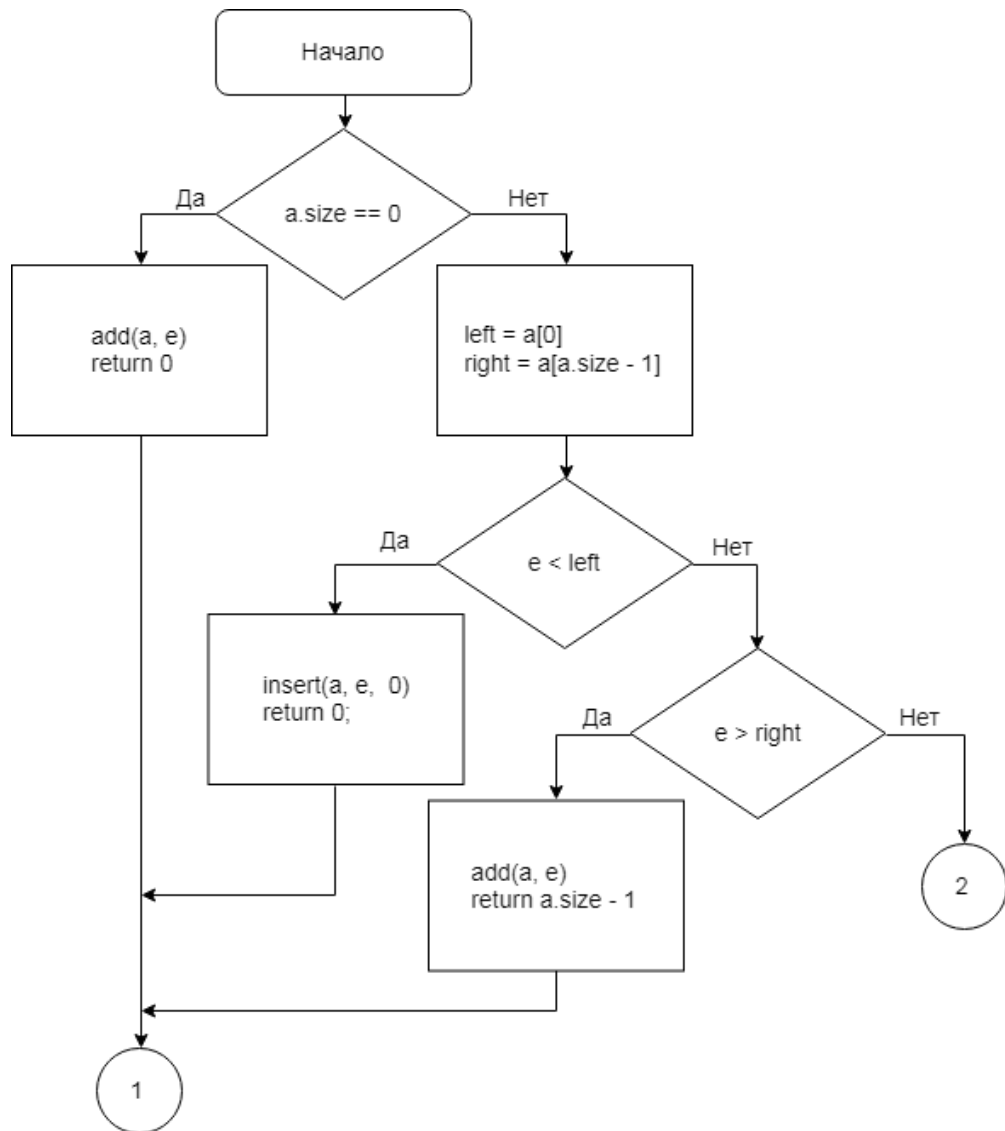


Рисунок 2.7 – Фрагмент алгоритма добавления элемента в отсортированный массив

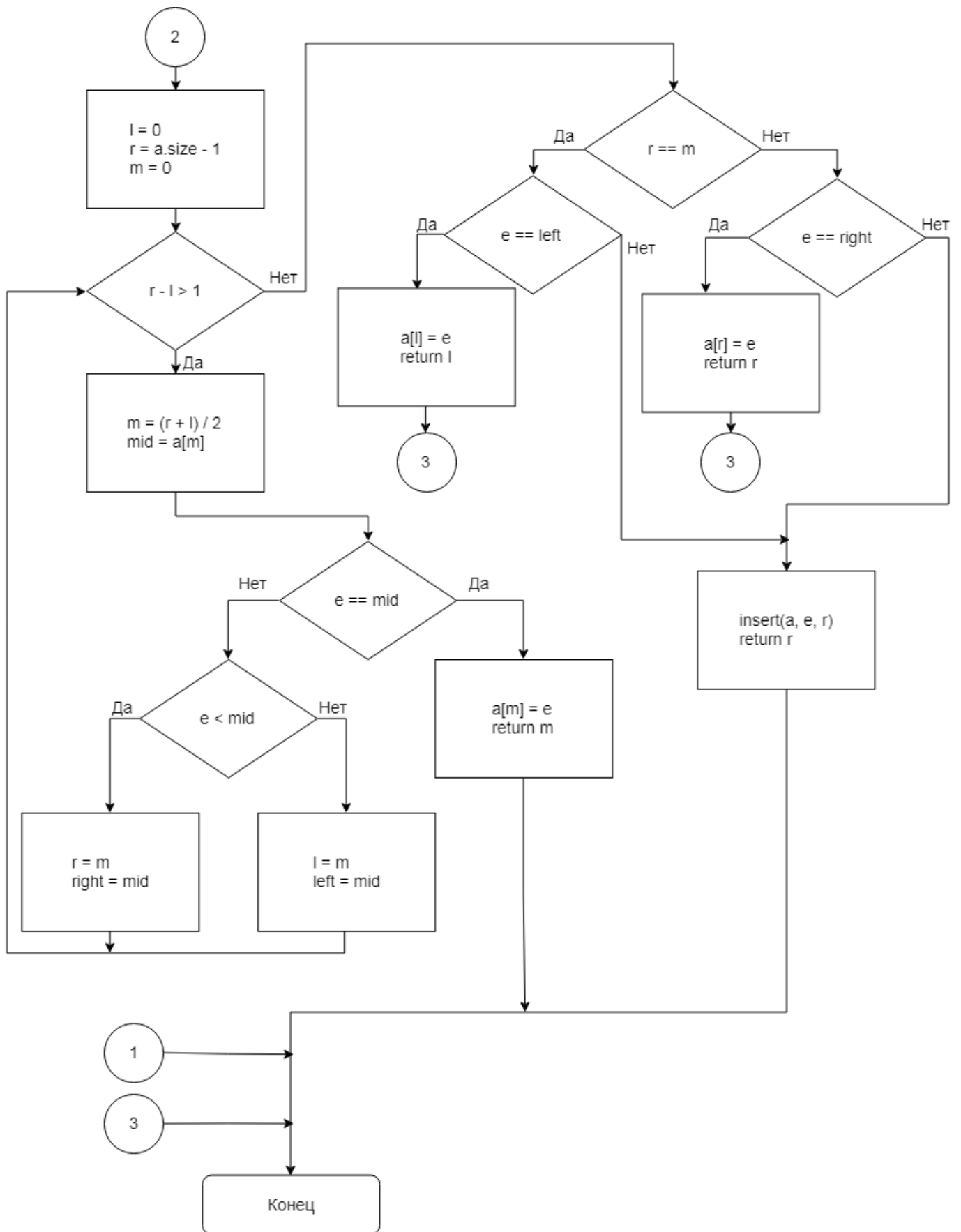


Рисунок 2.8 – Фрагмент алгоритма добавления элемента в отсортированный массив

Алгоритм функции добавления элемента в отсортированный массив изображен на рисунках 2.7-2.8. Он основан на алгоритме бинарного поиска, что существенно увеличивает скорость его работы. Средняя вычислительная сложность этого алгоритма – $O(\log n)$.

Так как добавление новой записи не всегда приводит к изменению видимой пользователю области данных, то значение, возвращаемое функцией добавления – индекс добавленного элемента, используется для определения необходимости обновления Вида.

За счет использования таких методов оптимизации, удалось добиться быстрой работы интерфейса программы.

2.6 Обоснование состава технических и программных средств

2.6.1 Выбор интегрированной среды разработки

В качестве интегрированной среды разработки (далее – IDE) была выбрана IDE Clion от компании JetBrains для разработки на языке C/C++. Данная среда разработки отвечает всем современным стандартам и совместима со всеми популярными компиляторами языка C, а также имеет встроенную поддержку системы сборки CMake.

2.6.2 Выбор TUI библиотеки

Для реализации качественного и удобного пользовательского интерфейса необходимо было реализовать поддержку управления как с помощью клавиатуры, так и с помощью мыши. Также, важным параметром являлась кроссплатформенная поддержка этого функционала. Так как стандартная библиотека языка C для работы с консольным окном(терминалом) не реализует обработку ввода с помощью мыши, то необходимо было использовать стороннюю библиотеку, поддерживающую данный функционал.

Выбор библиотеки был сделан в пользу PDcurses на основе проведенного сравнения самых популярных библиотек для создания TUI в консольном окне (табл. 2.2).

Таблица 2.2 – Таблица сравнения TUI библиотек

Название	ЯП	ОС	Последняя версия	Ссылка
ncurses	C, Ada	Linux	6.1, 27.01.2018	https://www.gnu.org/software/ncurses/ncurses.html
Newt	C	Linux	0.52.20, 17.07.2017	https://pagure.io/newt
dialog	C	Linux	7.11.2018	http://invisible-island.net/dialog/dialog.html
CDK	C, C++	Linux	1.273, 6.03.2018	http://invisible-island.net/cdk/cdk.html
PDCurses	C	Linux, DOS, OS/2, Win32, X11, SDL	3.6, 14.02.2018	https://pdcurses.sourceforge.io/
PDCurses (Bill-Gray fork)	C	Linux, DOS, OS/2, Win32, X11, SDL	4.0.3 Alpha, 30.1.2018	https://github.com/Bill-Gray/PDCurses
Turbo Vision	Pascal, C++	Linux, FreeBSD	2.2.1, 17.11.2016	http://tvision.sourceforge.net/

Из этой таблицы видно, что библиотека PDCurses имеет поддержку большого количества операционных систем, в том числе ОС Windows. К сожалению, по сравнению с некоторыми представленными в таблице библиоте-

ками, выбранная библиотека имеет высокий уровень абстракции. То есть реализует низкоуровневые операции и не имеет готовых компонентов, необходимых для реализации типового пользовательского интерфейса (Common User Access, CUA). Поэтому требовалось разработать библиотеку компонентов TUI с более низким уровнем абстракции и реализовать необходимые типовые элементы интерфейса (Label, Edit, Button и т.д.).

2.6.3 Выбор библиотеки коллекций (динамических структур)

В качестве библиотеки коллекций была выбрана библиотека Collections-C, так как она содержит реализации большинства стандартных структур (динамический массив, список, хэш-таблица и т.д.), а также сортировку данных в них. Еще одним преимуществом библиотеки является то, что она компилируется при помощи CMake, что существенно упростило интеграцию данной библиотеки в разрабатываемое приложение.

3 ИСПОЛЬЗОВАНИЕ ПРОГРАММЫ

3.1 Системные требования программы

Разработанное приложение не содержит высокопроизводительной графики, не производит ресурсоемкие вычисления, поэтому имеет невысокие системные требования. Одна запись в бинарном файле занимает 143 байта (см. рис. 2.2), следовательно, файл, состоящий, из 1000 записей будет занимать менее 143 Кбайт. Фактический размер исполняемого файла – менее 1 Мбайта.

Таблица 3.1 – Системные требования программы Manufactury

Процессор	500 MHz
Оперативная память	128 Мб RAM
Операционная система	Windows XP /Vista /7 /8
Свободное место на диске	15 Мб
Устройства взаимодействия с пользователем	Клавиатура и/или мышь

3.2 Загрузка и запуск программы

Для запуска программы, исполняемый файл программы для ОС Windows manufactury.exe должен быть загружен на диск компьютера. Запуск производится путем двойного нажатия на исполняемый файл. После запуска приложения, появится главное окно программы со стартовым сообщением (рис. 3.1).

На данном этапе программа запущена и готова к работе. Для начала работы можно создать новый файл или открыть уже существующий.



Рисунок 3.1 – Главное окно программы со стартовым сообщением

3.3 Тестирование программы

Интерфейс программы частично реализует стандарт IBM CUA и поддерживает управление с помощью клавиатуры и/или мыши.

Для переключения между элементами управления в режиме клавиатуры используются клавиша “Tab” (переключение на следующий элемент) и комбинация клавиш “Shift+Tab” (переключение на предыдущий элемент). Активация выбранного управляющего элемента происходит по нажатию клавиши “Enter”. Использование программы с помощью мыши не отличается от стандартных программ операционной системы Windows.

Главное меню программы содержит 3 раздела: «Файл», «Правка», «Инструменты». Все функции меню имеют назначенные сочетания горячих клавиш.

3.3.1 Раздел меню «Файл»

В разделе находятся функции программы, связанные с операциями над файлами и файловой системой: создание, открытие, сохранение, пересохранение (“Сохранить как ...”).

Чтобы создать новый файл выбираем Файл > Новый (Ctrl + N).

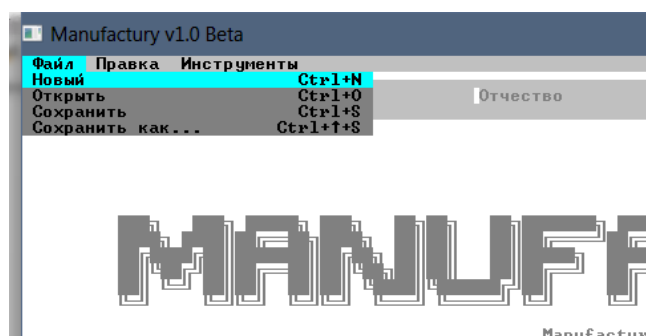
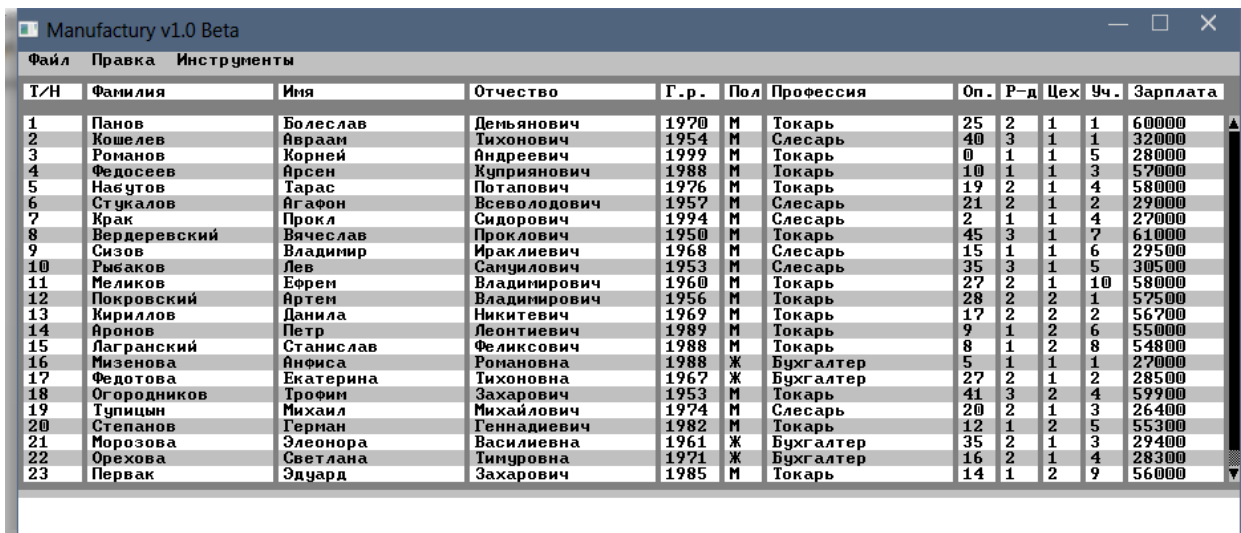


Рисунок 3.2 – Меню Файл > Новый

В главном окне программы отобразилась пустая таблица (рис. 3.3), а в памяти создавалась динамическая структура для хранения записей нового файла. На данном этапе физический файл еще не создан и информация не сохранена на диск.



Manufacture v1.0 Beta

Файл Правка Инструменты

Т/Н	Фамилия	Имя	Отчество	Г.р.	Пол	Профессия	Оп.	Р-д	Цех	Уч.	Зарплата
1	Панов	Болеслав	Демьянович	1970	М	Токарь	25	2	1	1	60000
2	Кошелев	Авраам	Тихонович	1954	М	Слесарь	40	3	1	1	32000
3	Романов	Корней	Андреевич	1999	М	Токарь	0	1	1	5	28000
4	Федосеев	Арсен	Куприянович	1988	М	Токарь	10	1	1	3	57000
5	Набутов	Тарас	Потапович	1976	М	Токарь	19	2	1	4	58000
6	Стукалов	Агафон	Всеволодович	1957	М	Слесарь	21	2	1	2	29000
7	Крак	Прокл	Сидорович	1994	М	Слесарь	2	1	1	4	27000
8	Вердеревский	Вячеслав	Проклович	1950	М	Токарь	45	3	1	7	61000
9	Сизов	Владимир	Ираклиевич	1968	М	Слесарь	15	1	1	6	29500
10	Рыбаков	Лев	Самуилович	1953	М	Слесарь	35	3	1	5	30500
11	Меликов	Ефрем	Владимирович	1960	М	Токарь	27	2	1	10	58000
12	Покровский	Артем	Владимирович	1956	М	Токарь	28	2	2	1	57500
13	Кириллов	Данила	Никитевич	1969	М	Токарь	17	2	2	2	56700
14	Аронов	Петр	Леонтиевич	1989	М	Токарь	9	1	2	6	55000
15	Лагранский	Станислав	Феликсович	1988	М	Токарь	8	1	2	8	54800
16	Мизенова	Анфиса	Романовна	1988	Ж	Бухгалтер	5	1	1	1	27000
17	Федотова	Екатерина	Тихоновна	1967	Ж	Бухгалтер	27	2	1	2	28500
18	Огородников	Трофим	Захарович	1953	М	Токарь	41	3	2	4	59900
19	Тупицын	Михаил	Михайлович	1974	М	Слесарь	20	2	1	3	26400
20	Степанов	Герман	Геннадиевич	1982	М	Токарь	12	1	2	5	55300
21	Морозова	Элеонора	Василиевна	1961	Ж	Бухгалтер	35	2	1	3	29400
22	Орехова	Светлана	Тинуровна	1971	Ж	Бухгалтер	16	2	1	4	28300
23	Первак	Эдуард	Захарович	1985	М	Токарь	14	1	2	9	56000

Рисунок 3.5 – Вид главного окна программы с таблицей, содержащей данные из файла

Если программе не удастся загрузить данные из файла (файл поврежден или не является файлом приложения Manufacture), то будет отображено сообщение об ошибке (рис. 3.6).

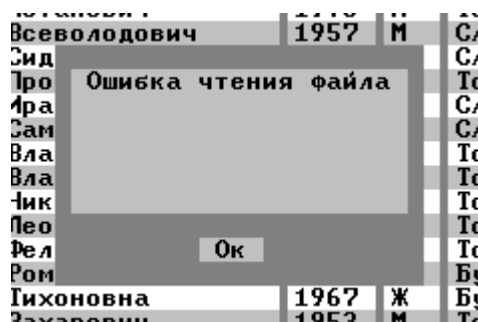


Рисунок 3.6 – Сообщение об ошибке чтения файла

Чтобы сохранить или пересохранить файл выбираем Файл > Сохранить (Ctrl + S) или Файл > Сохранить как ... (Ctrl + Shift + S) соответственно.

В диалоге сохранения файла (аналогичен диалогу открытия файла (рис. 3.4)) выбираем существующий файл для перезаписи или вводим свободное имя для создания нового файла.

Если сохранение файла не будет завершено успешно, то отобразится сообщение об ошибке записи, аналогичное сообщению об ошибке чтения файла, изображенному на рисунке 3.6.

Все операции этого раздела обеспечивают безопасность несохраненных данных в текущем файле. То есть, если при выполнении операций создания или открытия файла, активный (открытый) в данный момент файл будет иметь несохраненные изменения, то пользователь получит запрос на подтверждение сохранения текущих изменений или отмену операции (рис. 3.7).

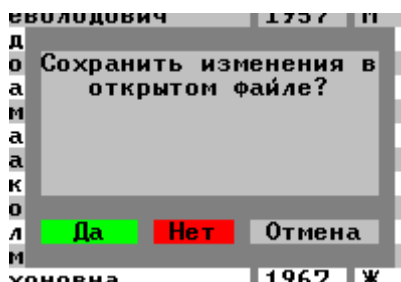


Рисунок 3.7 – Диалог подтверждения сохранения текущих изменений

3.3.2 Раздел меню «Правка»

В разделе находятся функции программы, связанные с редактированием записей открытого файла: добавление, удаление, изменение, поиск.

Чтобы добавить новую запись выбираем Правка > Добавить (Ctrl + I). На рисунке 3.8 изображен диалог добавления записи.

Рисунок 3.8 – Диалог добавления записи

Для удобства последовательного добавления записей, в поле “Табельный номер” помещается значение следующего порядкового номера. При этом если ввести в это поле номер, который уже занят, или “0”, то отобразится со-

общение об ошибке (рис. 3.9), и кнопка “Ок” деактивируется. После заполнения полей, подтверждаем добавление записи нажатием кнопки “Ок”. Запись успешно добавлена в таблицу и сохранена в памяти.

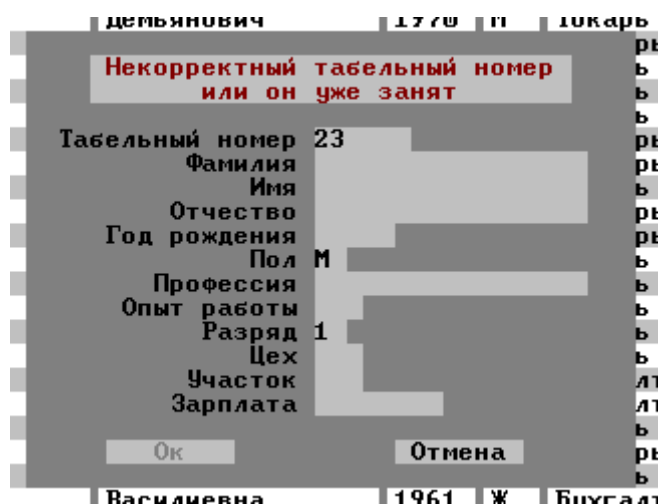


Рисунок 3.9 – Сообщение о некорректности табельного номера

Удалить запись можно двумя способами:

1. Выбираем Правка > Удалить (Ctrl + D) и в появившемся окне ввода табельного номера (рис. 3.10) вводим номер записи, которую нужно удалить. Если введен некорректный номер, то будет показано сообщение об отсутствии записи (рис. 3.11).
2. Выделяем нужную запись в таблице и используем сочетание клавиш Ctrl + D.

Если запись существует, то она будет удалена из памяти и из таблицы в главном окне.

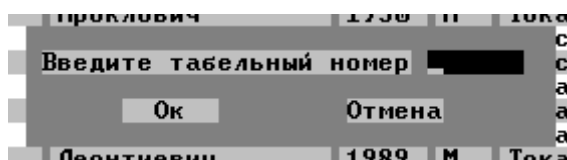


Рисунок 3.10 – Диалог ввода табельного номера

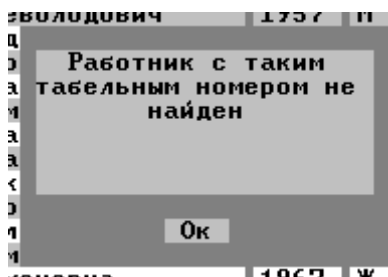


Рисунок 3.11 – Сообщение об отсутствии записи с введенным номером

Изменить запись так же можно несколькими способами:

1. Выбираем Правка > Изменить (Ctrl + E) и в окне, изображенном на рисунке 3.10, вводим номер записи которую необходимо изменить. Аналогично функции удаления записи, будет выведено сообщение об ошибке, если запись не найдена.
2. Выделяем нужную запись в таблице и используем сочетание клавиш Ctrl + E.

Если запись существует, то будет отображен диалог изменения записи (рис. 3.12) с заполненными полями.

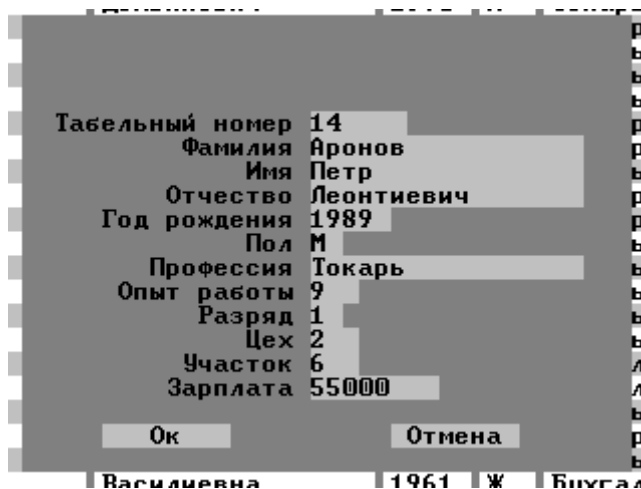


Рисунок 3.12 – Диалог изменения записи

Функция поиска реализована в виде фильтра для записей таблицы. Чтобы отфильтровать записи выбираем Правка > Найти (Ctrl + F). Появится окно ввода критерия для фильтрации, изображенное на рисунке 3.13. В этом окне можно выбрать поле, по которому будет проходить отбор и указать значение фильтра.

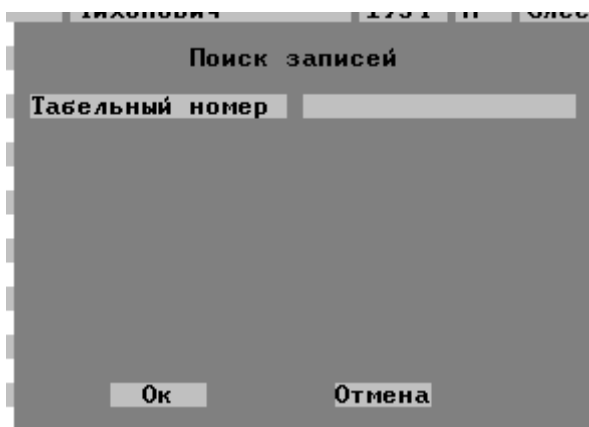


Рисунок 3.13 – Диалог поиска записей

После нажатия кнопки “Ок” в таблице останутся только те записи, которые удовлетворяют критерию поиска. Пример результата выполнения функции поиска изображен на рисунке 3.14.

Manufacture v1.0 Beta											
Файл Правка Инструменты											
Т/Н	Фамилия	Имя	Отчество	Г.р.	Пол	Профессия	Оп.	Р-д	Цех	Уч.	Зарплата
1	Панов	Болеслав	Демьянович	1970	М	Токарь	25	2	1	1	60000
3	Романов	Корней	Андреевич	1999	М	Токарь	0	1	1	5	28000
4	Федосеев	Арсен	Куприянович	1988	М	Токарь	10	1	1	3	57000
5	Набутов	Тарас	Потапович	1976	М	Токарь	19	2	1	4	58000
8	Вердеревский	Вячеслав	Проклович	1950	М	Токарь	45	3	1	7	61000
11	Меликов	Ефрем	Владимирович	1960	М	Токарь	27	2	1	10	58000
12	Покровский	Артём	Владимирович	1956	М	Токарь	28	2	2	1	57500
13	Кириллов	Данила	Никитевич	1969	М	Токарь	17	2	2	2	56700
14	Аронов	Петр	Леонтиевич	1989	М	Токарь	9	1	2	6	55000
15	Лагранский	Станислав	Феликсович	1988	М	Токарь	8	1	2	8	54800
18	Огородников	Трофим	Захарович	1953	М	Токарь	41	3	2	4	59900
20	Степанов	Герман	Геннадиевич	1982	М	Токарь	12	1	2	5	55300
23	Первак	Эдуард	Захарович	1985	М	Токарь	14	1	2	9	56000
24	Артемов	Самсон	Трофимович	1997	М	Токарь	1	1	2	10	45000
25	Макарин	Богдан	Анатолиевич	1969	М	Токарь	17	1	2	7	61000

Рисунок 3.14 – Результат поиска по критерию профессии “Токарь”

Для отмены текущего фильтра выбираем Правка > Отменить поиск (Alt + E). В таблице вновь отображены все записи текущего файла.

3.3.3 Раздел меню «Инструменты»

Чтобы создать отчет о количестве сотрудников определенной профессии с группировкой по разрядам и опыту работы выбираем Инструменты > Создать отчет (Ctrl + R). В появившемся окне (рис. 3.15) вводим профессию, по которой необходимо создать отчет.

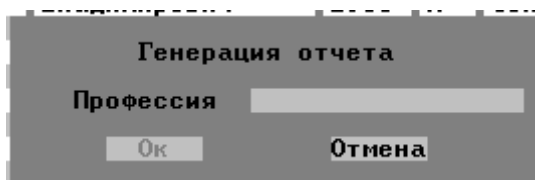


Рисунок 3.15 – Диалог создания отчета

После подтверждения ввода нажатием кнопки “Ок”, появится окно предварительного просмотра отчета. Вид окна предпросмотра изображен на рисунке 3.16.

Стаж работы	Количество рабочих по разрядам			Всего
	1	2	3	
До 6	2	0	0	2
С 6 до 11	3	0	0	3
С 11 до 16	2	0	0	2
С 16 до 21	1	2	0	3
С 21 до 25	0	1	0	1
Свыше 25	0	2	2	4

Сохранить Удалить

Рисунок 3.16 – Окно предпросмотра сгенерированного отчета

Чтобы сохранить отчет в текстовый файл, нажимаем на кнопку “Сохранить”. Содержимое файла отчета показано на рисунке 3.17.

report.txt

Стаж работы	Количество рабочих по разрядам			Всего
	1	2	3	
До 6	2	0	0	2
С 6 до 11	3	0	0	3
С 11 до 16	2	0	0	2
С 16 до 21	1	2	0	3
С 21 до 25	0	1	0	1
Свыше 25	0	2	2	4

Рисунок 3.17 – Содержимое файла отчета

Если нажать кнопку “Удалить”, то отчет не будет сохранен файл, диалог будет закрыт.

3.3.4 Функции табличного редактора

Главное окно представляет собой таблицу, содержащую в клетках данные записей файла. Кроме функции просмотра, этот элемент интерфейса реализует функции сортировки и редактирования записей.

Чтобы отсортировать записи в таблице, нажимаем на заголовок столбца, по которому необходимо провести сортировку (ЛКМ – по возрастанию, ПКМ – по убыванию). Результат сортировки изображен на рисунке 3.18.

T/N	Фамилия	Имя	Отчество	Г.р.	Пол	Профессия	Оп.	Р-д	Цех	Уч.	Зарплата
14	Аронов	Петр	Леонтиевич	1989	М	Токарь	9	1	2	6	55000
24	Артемов	Самсон	Трофимович	1997	М	Токарь	1	1	2	10	45000
8	Вердереvский	Вячеслав	Проклович	1950	М	Токарь	45	3	1	7	61000
13	Кириллов	Данила	Никитевич	1969	М	Токарь	17	2	2	2	56700
2	Кошелев	Авраам	Тихонович	1954	М	Слесарь	40	3	1	1	32000
7	Крак	Прокл	Сидорович	1994	М	Слесарь	2	1	1	4	27000
15	Лагранский	Станислав	Феликсович	1988	М	Токарь	8	1	2	8	54800
25	Макарин	Богдан	Анатолиевич	1969	М	Токарь	17	1	2	7	61000
11	Меликов	Ефрем	Владимирович	1960	М	Токарь	27	2	1	10	58000
16	Мизенова	Анфиса	Романовна	1988	Ж	Бухгалтер	5	1	1	1	27000
21	Морозова	Элеонора	Василиевна	1961	Ж	Бухгалтер	35	2	1	3	29400
5	Набутов	Тарас	Потапович	1976	М	Токарь	19	2	1	4	58000
18	Огородников	Трофим	Захарович	1953	М	Токарь	41	3	2	4	59900
22	Орехова	Светлана	Тинуровна	1971	Ж	Бухгалтер	16	2	1	4	28300
1	Панов	Болеслав	Демьянович	1970	М	Токарь	25	2	1	1	60000
23	Первак	Эдуард	Захарович	1985	М	Токарь	14	1	2	9	56000
12	Покровский	Артем	Владимирович	1956	М	Токарь	28	2	2	1	57500
3	Романов	Корней	Андреевич	1999	М	Токарь	0	1	1	5	28000
10	Рыбаков	Лев	Самуилович	1953	М	Слесарь	35	3	1	5	30500
9	Сизов	Владимир	Ираклиевич	1968	М	Слесарь	15	1	1	6	29500
20	Степанов	Герман	Геннадиевич	1982	М	Токарь	12	1	2	5	55300
6	Стукалов	Агафон	Всеволодович	1957	М	Слесарь	21	2	1	2	29000
19	Тупицын	Михаил	Михайлович	1974	М	Слесарь	20	2	1	3	26400

Рисунок 3.18 – Результат сортировки по столбцу “Фамилия”

Для редактирования записей прямо в таблице, достаточно выделить мышью ячейку, которую необходимо изменить, и ввести новое значение поля данной записи.

ВЫВОДЫ

Для достижения поставленной цели были изучены принципы проектирования пользовательских интерфейсов в приложениях для операционных систем с графическим интерфейсом, проведено сравнение существующих библиотек, предназначенных для создания интерфейсов в консольном окне. По итогам сравнительного анализа, был сделан вывод о необходимости разработки дополнительной библиотеки. Таким образом, была разработана библиотека компонентов текстового пользовательского интерфейса (TUI), реализующая основные элементы: Label, Button, Select, ScrollBar, Edit. Также, были изучены оптимальные способы хранения информации, позволяющие эффективно обрабатывать данные о сотрудниках предприятия, и разработана модель данных на основе динамической структуры ArrayList. На основе разработанной библиотеки и модели данных, было реализовано приложение со стандартным пользовательским интерфейсом, имеющее функции для создания, хранения и обработки информации о составе работников предприятия. Также был реализован дополнительный элемент интерфейса – ColumnLabel, позволяющий управлять сортировкой записей в главном окне программы.

В заключении можно отметить, что все поставленные задачи были выполнены. Приложение «Manufactury», позволяющее вести учет и мониторинг состава сотрудников предприятия, а также создавать отчеты с группировкой по разрядам и стажу работы, было разработано и полностью протестировано. Созданная программа выполняет все функции корректно. То есть цель курсового проекта успешно достигнута.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. curses [Электронный ресурс]. - Режим доступа: <https://ru.wikipedia.org/wiki/Curses>. - Curses. - (Дата обращения: 02.10.2018)
2. IBM Common User Access [Электронный ресурс]. - Режим доступа: https://ru.wikipedia.org/wiki/IBM_Common_User_Access. - Common User Access (типовой пользовательский интерфейс, CUA). - (Дата обращения: 02.11.2018).
3. Microsoft Patterns & Practices Team Microsoft® Application Architecture Guide (Patterns & Practices). - 2-ое изд. - Microsoft Corporation, 2009. - 524 с.
4. Text-based user interface [Электронный ресурс]. - Режим доступа: https://en.wikipedia.org/wiki/Text-based_user_interface. - Text-based user interface (TUI). - (Дата обращения: 10.10.2018)
5. Бодягин, И.Д. Model-View-Presenter и сопутствующие паттерны [Электронный ресурс] // RSDN Magazine. - 2006. - №2. - Режим доступа: <http://rsdn.org/article/patterns/ModelViewPresenter.xml>. - (Дата обращения: 13.11.2018)
6. Керниган, Б.У. Язык программирования С: учеб. Пособие / Б. У. Керниган, Д. М. Ритчи.- 2-ое изд. - М.: Изд-во Вильямс, 2015. - 304 с.

Приложение А. Исходный код программы

Файл main.c:

```
#include <stdio.h>
#include <curses.h>
#include <panel.h>
#include <src/log.h>
#include <tui/styles.h>
#include <tui/dialog.h>
#include <edit/edit_view.h>
#include <model/data_source.h>
#include <edit/id_input_dialog.h>
#include <report/report_view.h>
#include <report/report_param_dialog.h>
#include <main/filter_dialog.h>
#include "global.h"
#include "main/main_view.h"
#include "main/main_presenter.h"
#include "version.h"

#ifdef DEBUG_LOG
    #define LOG_FILE "log.txt"
#endif

void initPDCurses() {
    //инициализация главного окна
    initscr();
    //режим распознавания каждого нажатия клавиши без ожидания Enter
    cbreak();
    //отключение отображения введенного символа
    noecho();
    //отключение курсора
    curs_set(0);
    //неблокирующий ввод
    nodelay(stdscr, TRUE);
    //распознавание функциональных кнопок клавиатуры(F1, ...)
    keypad(stdscr, TRUE);
    //сохранение модификаторов нажатых клавиш при чтении(Ctrl, Alt, Shift, ...)
    PDC_save_key_modifiers(TRUE);
    //поддержка цвета
    start_color();
    //поддержка мыши
    mouse_set(ALL_MOUSE_EVENTS);
    //установка заголовка окна программы
    PDC_set_title("Manufactury v" MANUFACTURY_VERSION);
}

void endCurses() {
    //завершение работы curses
    endwin();
}

int main(int argc, char** argv) {
#ifdef DEBUG_LOG
    FILE* log = fopen(LOG_FILE, "wt");
    log_set_fp(log);
#endif
    initPDCurses();
    InitStyle();
    InitHotKeyHandler(NULL);
    InitDataSource();
    InitMainView();
    InitDialogsLayouts(dialogBackground);
    InitDataDialog(dialogBackground);
    InitIdInputDialog(dialogBackground);
    InitReportParamDialog(dialogBackground);
    InitFilterDialog(dialogBackground);
    InitReportView(dialogBackground);
    InitApplication(NULL);
    StartControl();
    endCurses();
#ifdef DEBUG_LOG
    fclose(log);
#endif
}
```

```
#endif
}
```

Файл main_presenter.h:

```
#ifndef MANUFACTURY_MAIN_PRESENTER_H
#define MANUFACTURY_MAIN_PRESENTER_H

#include <model/data_types.h>
#include <model/comparator.h>
#include <tui/layout.h>
#include <tui/interactive_panel.h>
#include "global.h"

void InitApplication(char* param);

void FileNew(void);

void FileOpen(void);

void FileSave(void);

void FileSaveAs(void);

void EditFind(void);

void EditCancelFind(void);

void EditAdd(void);

void EditDelete(void);

void EditChange(void);

void ToolsExportCSV(void);

void ToolsCreateReport(void);

void TableOnScrollDown(ScrollType scrollType);

void TableOnScrollUp(ScrollType scrollType);

void SortData(int fieldId, SortDirection direction);

void EditEntry(Component* handle);

void OnIdButtonMouseClicked(InteractivePanel* handle, MEVENT event);

void OnIdButtonKeyClick(Component* handle, int key, unsigned long modifiers);

void ColumnChanged(int fieldId);

void ChangeSurname(Component* handle);

void ChangeName(Component* handle);

void ChangePatronymic(Component* handle);

void ChangeYOB(Component* handle);

void ChangeGender(Component* handle);

void ChangeProfession(Component* handle);

void ChangeExperience(Component* handle);

void ChangeClass(Component* handle);

void ChangeDepartment(Component* handle);

void ChangePlot(Component* handle);

void ChangeSalary(Component* handle);

void EntryAdded(Employee* e);
```

```
void EntryChanged(Employee* e);

#endif //MANUFACTURY_MAIN_PRESENTER_H
```

Файл main_presenter.c:

```
#include <model/data_types.h>
#include <src/log.h>
#include <model/data_source.h>
#include <edit/edit_view.h>
#include <model/array_ext.h>
#include <tui/winapi_bridge.h>
#include <tui/dialog.h>
#include <edit/edit_presenter.h>
#include <tui/component/edit.h>
#include <tui/component/select.h>
#include <minmax.h>
#include <tui/component/button.h>
#include <edit/id_input_dialog.h>
#include <report/report_presenter.h>
#include <model/predicate.h>
#include "main_presenter.h"
#include "main_view.h"
#include "filter_dialog.h"

Array* sorted = NULL;
int pos = 0;
int pageSize = 23;
int sortField = 0;
SortDirection sortDirection = ASC;
Comparator* comparators[FIELDS_COUNT];

/*
 * true - Работа с данными, сохраненными в существующий файл
 * false - Работа с данными, не сохраненными ни в какой файл
 */
bool currentOpened = false;

char currentFileName[PATH_MAX] = { '\0' };
bool currentChanged = false;
bool needSorting = false;

int (* predicates[FIELDS_COUNT]) (void* value, void* e);
void* predicateParameter = NULL;
int (* predicate)(void* value, void* e);

void updateData(Array* data) {
    if (data == NULL) {
        if (sorted != NULL) {
            array_destroy(sorted);
            sorted = NULL;
            ShowStarter();
        }
    } else {
        if (sorted == NULL) {
            ShowTable();
        } else {
            array_destroy(sorted);
            sorted = NULL;
        }
        array_copy_shallow(data, &sorted);
        sortField = 0;
        SetData(sorted);
        ResetActiveColumnLabel();
    }
}

void setCurrentFile(char* fileName) {
    currentChanged = false;
    if (fileName == NULL) {
        currentOpened = false;
    } else {
        currentOpened = true;
        strcpy(currentFileName, fileName);
    }
}
```

```

void newFile(void) {
    setCurrentFile(NULL);
    wipeData();
    updateData(GetEmployees());
}

void deleteEntry(Employee* e) {
    currentChanged = true;
    RemoveEmployee(e);
    size_t removePos = 0;
    array_index_of(sorted, e, &removePos);
    array_remove(sorted, e, NULL);
    if (removePos >= pos && removePos < pos + pageSize) {
        SetPos(pos);
    }
}

void deleteById(int id) {
    Employee* e = GetEmployee(id);
    if (e == NULL) {
        ShowMessageDialog(L"Работник с таким табельным номером не найден", NULL);
    } else {
        deleteEntry(e);
    }
}

void changeById(int id) {
    Employee* e = GetEmployee(id);
    if (e == NULL) {
        ShowMessageDialog(L"Работник с таким табельным номером не найден", NULL);
    } else {
        ShowChangeDialog(e);
    }
}

bool filterData(const void* e) {
    return predicate(predicateParameter, (void*) e) == 0;
}

void filterAll(int fieldId, wchar_t* value) {
    bool mem = false;
    bool filter = true;
    switch (fieldId) {
        case FIELD_ID:
        case FIELD_YOB:
        case FIELD_EXPERIENCE:
        case FIELD_DEPARTMENT:
        case FIELD_PLOT:
        case FIELD_SALARY:
            predicateParameter = malloc(sizeof(int));
            mem = true;
            *((int*) predicateParameter) = parseInt(value);
            if (*((int*) predicateParameter) == -1) {
                filter = false;
            }
            break;
        case FIELD_GENDER:
            predicateParameter = malloc(sizeof(bool));
            mem = true;
            switch (value[0]) {
                case L'M':
                    *((bool*) predicateParameter) = true;
                    break;
                case L'Ж':
                    *((bool*) predicateParameter) = false;
                    break;
                default:
                    filter = false;
                    break;
            }
            break;
        case FIELD_CLASS:
            predicateParameter = malloc(sizeof(ProfClass));
            mem = true;
            switch (value[0]) {

```

```

        case L'1':
            *((ProfClass*) predicateParameter) = FIRST;
            break;
        case L'2':
            *((ProfClass*) predicateParameter) = SECOND;
            break;
        case L'3':
            *((ProfClass*) predicateParameter) = THIRD;
            break;
        default:
            filter = false;
            break;
    }
    break;
default:
    predicateParameter = value;
    break;
}
if (filter) {
    predicate = predicates[fieldId];
    Array* filtered;
    array_copy_shallow(GetEmployees(), &filtered);
    array_filter_mut(filtered, filterData);
    if (array_size(filtered) == 0) {
        ShowFilterError();
    } else {
        updateData(filtered);
    }
} else {
    ShowFilterError();
}
if (mem) {
    free(predicateParameter);
    predicateParameter = NULL;
}
}

void InitApplication(char* param) {
    comparators[FIELD_ID] = CreateComparator(EmployeeIdComparator, EmployeeIdComparatorReversed);
    comparators[FIELD_SURNAME] = CreateComparator(EmployeeSurnameComparator, EmployeeSurnameComparatorReversed);
    comparators[FIELD_NAME] = CreateComparator(EmployeeNameComparator, EmployeeNameComparatorReversed);
    comparators[FIELD_PATRONYMIC] = CreateComparator(EmployeePatronymicComparator, EmployeePatronymicComparatorReversed);
    comparators[FIELD_YOB] = CreateComparator(EmployeeYOBComparator, EmployeeYOBComparatorReversed);
    comparators[FIELD_GENDER] = CreateComparator(EmployeeGenderComparator, EmployeeGenderComparatorReversed);
    comparators[FIELD_PROFESSION] = CreateComparator(EmployeeProfessionComparator, EmployeeProfessionComparatorReversed);
    comparators[FIELD_EXPERIENCE] = CreateComparator(EmployeeExperienceComparator, EmployeeExperienceComparatorReversed);
    comparators[FIELD_CLASS] = CreateComparator(EmployeeClassComparator, EmployeeClassComparatorReversed);
    comparators[FIELD_DEPARTMENT] = CreateComparator(EmployeeDepartmentComparator, EmployeeDepartmentComparatorReversed);
    comparators[FIELD_PLOT] = CreateComparator(EmployeePlotComparator, EmployeePlotComparatorReversed);
    comparators[FIELD_SALARY] = CreateComparator(EmployeeSalaryComparator, EmployeeSalaryComparatorReversed);

    predicates[FIELD_ID] = (int (*)(void*, void*)) EmployeeIdPredicate;
    predicates[FIELD_SURNAME] = (int (*)(void*, void*)) EmployeeSurnamePredicate;
    predicates[FIELD_NAME] = (int (*)(void*, void*)) EmployeeNamePredicate;
    predicates[FIELD_PATRONYMIC] = (int (*)(void*, void*)) EmployeePatronymicPredicate;
    predicates[FIELD_YOB] = (int (*)(void*, void*)) EmployeeYOBPredicate;
    predicates[FIELD_GENDER] = (int (*)(void*, void*)) EmployeeGenderPredicate;
    predicates[FIELD_PROFESSION] = (int (*)(void*, void*)) EmployeeProfessionPredicate;
    predicates[FIELD_EXPERIENCE] = (int (*)(void*, void*)) EmployeeExperiencePredicate;
    predicates[FIELD_CLASS] = (int (*)(void*, void*)) EmployeeClassPredicate;
    predicates[FIELD_DEPARTMENT] = (int (*)(void*, void*)) EmployeeDepartmentPredicate;
    predicates[FIELD_PLOT] = (int (*)(void*, void*)) EmployeePlotPredicate;
    predicates[FIELD_SALARY] = (int (*)(void*, void*)) EmployeeSalaryPredicate;

    currentFileName[0] = '\0';
}

```

```

        if (param == NULL) {
            ShowStarter();
        }
    }

void newSaveChanges(void) {
    if (currentOpened) {
        if (WriteFile(currentFileName)) {
            newFile();
        } else {
            ShowFileWriteError();
        }
    } else {
        char fileName[PATH_MAX] = { '\0' };
        if (ShowSaveFileDialog(fileName, PATH_MAX)) {
            if (WriteFile(fileName)) {
                newFile();
            } else {
                ShowFileWriteError();
            }
        }
    }
}

void newDiscardChanges(void) {
    newFile();
}

void openSaveChanges(void) {
    char fileName[PATH_MAX] = { '\0' };
    if (currentOpened) {
        if (WriteFile(currentFileName)) {
            if (ShowOpenFileDialog(fileName, PATH_MAX)) {
                if (ReadFile(fileName)) {
                    setCurrentFile(fileName);
                } else {
                    ShowFileWriteError();
                }
            }
        } else {
            ShowFileWriteError();
        }
    } else {
        if (ShowSaveFileDialog(fileName, PATH_MAX)) {
            if (WriteFile(fileName)) {
                setCurrentFile(fileName);
                if (ShowOpenFileDialog(fileName, PATH_MAX)) {
                    if (ReadFile(fileName)) {
                        setCurrentFile(fileName);
                    } else {
                        ShowFileWriteError();
                    }
                }
            } else {
                ShowFileWriteError();
            }
        }
    }
}

void openDiscardChanges(void) {
    char fileName[PATH_MAX] = { '\0' };
    if (ShowOpenFileDialog(fileName, PATH_MAX)) {
        if (ReadFile(fileName)) {
            setCurrentFile(fileName);
            updateData(GetEmployees());
        } else {
            ShowFileReadError();
        }
    }
}

void FileNew(void) {
    if (currentChanged) {

```

```

        ShowConfirmationDialog(L"Сохранить изменения в открытом файле?", newSaveChanges, newDis-
cardChanges, NULL); //newSaveChanges, newDiscardChanges
    } else {
        newFile();
    }
}

void FileOpen(void) {
    if (currentChanged) {
        ShowConfirmationDialog(L"Сохранить изменения в открытом файле?", openSaveChanges, openDis-
cardChanges, NULL); //openSaveChanges, openDiscardChanges
    } else {
        char fileName[PATH_MAX] = { '\0' };
        if (ShowOpenFileDialog(fileName, PATH_MAX)) {
            if (ReadFile(fileName)) {
                setCurrentFile(fileName);
                updateData(GetEmployees());
            } else {
                ShowFileReadError();
            }
        }
    }
}

void FileSave(void) {
    if (currentChanged) {
        if (currentOpened) {
            WriteFile(currentFileName);
            currentChanged = false;
        } else {
            char fileName[PATH_MAX] = { '\0' };
            if (ShowSaveFileDialog(fileName, PATH_MAX)) {
                if (WriteFile(fileName)) {
                    setCurrentFile(fileName);
                } else {
                    ShowFileWriteError();
                }
            }
        }
    }
}

void FileSaveAs(void) {
    char fileName[PATH_MAX] = { '\0' };
    if (ShowSaveFileDialog(fileName, PATH_MAX)) { //Имя для файла выбрано
        if (currentOpened) {
            if (strcmp(currentFileName, fileName) == 0) { //Выбран тот же файл
                if (WriteFile(currentFileName)) {
                    currentChanged = false;
                } else {
                    ShowFileWriteError();
                }
            } else { //Выбран другой файл
                if (WriteFile(fileName)) {
                    setCurrentFile(fileName);
                } else {
                    ShowFileWriteError();
                }
            }
        } else {
            if (WriteFile(currentFileName)) {
                setCurrentFile(fileName);
            } else {
                ShowFileWriteError();
            }
        }
    }
}

void EditFind(void) {
    if (sorted == NULL) return;
    ShowFilterDialog(filterAll);
}

void EditCancelFind(void) {
    if (sorted == NULL) return;
}

```

```

        updateData(GetEmployees());
    }

    void EditAdd(void) {
        if (sorted == NULL) return;
        ShowAddDialog();
    }

    void EditDelete(void) {
        if (sorted == NULL) return;
        Component* component = GetFocusedComponent();
        if (component != NULL && component->custom != NULL) {
            deleteEntry(component->custom);
        } else {
            ShowIdInputDialog(deleteById);
        }
    }

    void EditChange(void) {
        if (sorted == NULL) return;
        Component* component = GetFocusedComponent();
        if (component != NULL && component->custom != NULL) {
            EditEntry(component);
        } else {
            ShowIdInputDialog(changeById);
        }
    }

    void ToolsExportCSV(void) {
    }

    void ToolsCreateReport(void) {
        if (sorted == NULL) return;
        ShowReportDialog();
    }

    void TableOnScrollDown(ScrollType scrollType) {
        if (sorted == NULL) return;
        int step = 1;
        if (scrollType == PAGE) {
            step = pageSize;
        }
        if (array_size(sorted) == 0 || pos == array_size(sorted) - 1) return;
        if (pos + step > array_size(sorted) - 1) {
            pos = (int) (array_size(sorted) - 1);
        } else {
            pos += step;
        }
        SetPos(pos);
    }

    void TableOnScrollUp(ScrollType scrollType) {
        if (sorted == NULL) return;
        int step = 1;
        if (scrollType == PAGE) {
            step = pageSize;
        }
        if (array_size(sorted) == 0 || pos == 0) return;
        if (pos - step < 0) {
            pos = 0;
        } else {
            pos -= step;
        }
        SetPos(pos);
    }

    void SortData(int fieldId, SortDirection direction) {
        if (sorted == NULL) return;
        if (direction == NONE) return;
        if (sortField == fieldId) { //признак сортировки не изменился
            if (sortDirection != direction) { //сортировка не требуется, достаточно перевернуть массив
                sortDirection = direction;
                array_reverse(sorted);
                pos = 0;
                SetPos(0);
            }
        }
    }

```



```

    }
    } else { //признак сортировки изменился, требуется сортировка
        sortField = fieldId;
        sortDirection = direction;
        if (sortDirection == ASC) {
            array_sort(sorted, (int (*)(const void*, const void*)) comparators[sortField]->com-
pare);
        } else {
            array_sort(sorted, (int (*)(const void*, const void*)) comparators[sortField]->com-
pareReversed);
        }
        pos = 0;
        SetPos(0);
    }
}

void EditEntry(Component* handle) {
    ShowChangeDialog(handle->custom);
}

void OnIdButtonMouseClicked(InteractivePanel* handle, MEVENT event) {
    if (event.bstate & BUTTON1_DOUBLE_CLICKED) {
        FocusSingleComponent(NULL);
        EditEntry(handle->holder);
    }
}

void OnIdButtonKeyClick(Component* handle, int key, unsigned long modifiers) {
    if (key == KEY_DELETE) {
        FocusSingleComponent(NULL);
        deleteEntry(handle->custom);
    } else if (key == KEY_ENTER) {
        FocusSingleComponent(NULL);
        EditEntry(handle);
    }
}

void ChangeSurname(Component* handle) {
    Employee* e = handle->custom;
    wchar_t* surname = EditGetValue(handle);
    if (wcscmp(surname, e->surname) != 0) {
        EmployeeSetSurname(e, surname);
        ColumnChanged(FIELD_SURNAME);
    }
}

void ChangeName(Component* handle) {
    Employee* e = handle->custom;
    wchar_t* name = EditGetValue(handle);
    if (wcscmp(name, e->name) != 0) {
        EmployeeSetName(e, name);
        ColumnChanged(FIELD_NAME);
    }
}

void ChangePatronymic(Component* handle) {
    Employee* e = handle->custom;
    wchar_t* patronymic = EditGetValue(handle);
    if (wcscmp(patronymic, e->patronymic) != 0) {
        EmployeeSetPatronymic(e, patronymic);
        ColumnChanged(FIELD_PATRONYMIC);
    }
}

void ChangeYOB(Component* handle) {
    Employee* e = handle->custom;
    int yob = parseInt(EditGetValue(handle));
    if (yob != e->yob) {
        e->yob = (short) yob;
        ColumnChanged(FIELD_YOB);
    }
}

void ChangeGender(Component* handle) {
    Employee* e = handle->custom;
    bool gender = ValueOfGender(SelectGetValue(handle));

```

```

        if (gender != e->gender) {
            e->gender = gender;
            ColumnChanged(FIELD_GENDER);
        }
    }

    void ChangeProfession(Component* handle) {
        Employee* e = handle->custom;
        wchar_t* profession = EditGetValue(handle);
        if (wcscmp(profession, e->profession) != 0) {
            EmployeeSetProfession(e, profession);
            ColumnChanged(FIELD_PROFESSION);
        }
    }

    void ChangeExperience(Component* handle) {
        Employee* e = handle->custom;
        int experience = parseInt(EditGetValue(handle));
        if (experience != e->experience) {
            e->experience = (char) experience;
            ColumnChanged(FIELD_EXPERIENCE);
        }
    }

    void ChangeClass(Component* handle) {
        Employee* e = handle->custom;
        ProfClass class = ValueOfProfClass(SelectGetValue(handle));
        if (class != e->class) {
            e->class = class;
            ColumnChanged(FIELD_CLASS);
        }
    }

    void ChangeDepartment(Component* handle) {
        Employee* e = handle->custom;
        int department = parseInt(EditGetValue(handle));
        if (department != e->department) {
            e->department = (char) department;
            ColumnChanged(FIELD_DEPARTMENT);
        }
    }

    void ChangePlot(Component* handle) {
        Employee* e = handle->custom;
        int plot = parseInt(EditGetValue(handle));
        if (plot != e->plot) {
            e->plot = (char) plot;
            ColumnChanged(FIELD_PLOT);
        }
    }

    void ChangeSalary(Component* handle) {
        Employee* e = handle->custom;
        int salary = parseInt(EditGetValue(handle));
        if (salary != e->salary) {
            e->salary = (char) salary;
            ColumnChanged(FIELD_SALARY);
        }
    }

    void ColumnChanged(int fieldId) {
        currentChanged = true;
        if (sortField == fieldId) {
            needSorting = true;
        }
    }

    void EntryAdded(Employee* e) {
        currentChanged = true;
        int updated = 0;
        if (sortDirection == ASC) {
            updated = array_sorted_add(sorted, e, (int (*)(void*, void*)) comparators[sortField]->compare, NULL);
        } else {
            updated = array_sorted_add(sorted, e, (int (*)(void*, void*)) comparators[sortField]->compareReversed, NULL);
        }
    }

```

```

    }
    if (updated >= pos && updated < pos + pageSize) {
        SetPos(pos);
    }
}

void EntryChanged(Employee* e) {
    if (needSorting) {
        needSorting = false;
        size_t oldPos;
        if (array_index_of(sorted, e, &oldPos) != CC_OK) oldPos = 0;
        if (sortDirection == ASC) {
            array_sort(sorted, (int (*)(const void*, const void*)) comparators[sortField]->compare);
        } else {
            array_sort(sorted, (int (*)(const void*, const void*)) comparators[sortField]->compareReversed);
        }
        size_t newPos;
        if (array_index_of(sorted, e, &newPos) != CC_OK) newPos = 0;
        if (newPos != oldPos) {
            pos = (int) (newPos - max((oldPos - pos), 0));
            SetPos(pos);
        }
    } else {
        ShowEntryChanges(e);
    }
}

```

Файл main_view.h:

```

#ifndef MANUFACTURY_MAIN_VIEW_H
#define MANUFACTURY_MAIN_VIEW_H

#include <global.h>
#include <array.h>
#include <model/data_types.h>

void InitMainView(void);
void StartControl(void);
void SetPos(int pos);
void SetData(Array* data);
void ShowStarter(void);
void ShowTable(void);
void ResetActiveColumnLabel(void);
void ShowFileReadError(void);
void ShowFileWriteError(void);
void ShowEntryChanges(Employee* e);
void ShowFilterError(void);

#endif //MANUFACTURY_MAIN_VIEW_H

```

Файл main_view.c:

```

#include <curses.h>
#include <panel.h>
#include <src/log.h>
#include <minmax.h>
#include <model/data_types.h>
#include <model/filter.h>
#include <tui/layout.h>
#include <tui/styles.h>
#include <tui/hotkey.h>
#include <tui/dialog.h>
#include <tui/winapi_bridge.h>
#include <version.h>
#include "main_view.h"
#include "main_presenter.h"
#include "edit/edit_view.h"
#include "logo.h"

#define HEIGHT 30
#define WIDTH 115

```

```

#define LOGO_Y 10
#define VERSION_Y 17
#define HELP_Y 19

#define MAX_TABLE_SIZE 23
#define COLUMNS_COUNT FIELDS_COUNT
#define TABLE_LABEL_Y 2
#define TABLE_BODY_START_Y TABLE_LABEL_Y + 2

#define COL_ID_X 1
#define COL_ID_WIDTH 5
#define COL_SURNAME_X COL_ID_X+1+COL_ID_WIDTH
#define COL_SURNAME_WIDTH 17
#define COL_NAME_X COL_SURNAME_X+1+COL_SURNAME_WIDTH
#define COL_NAME_WIDTH 17
#define COL_PATRONYMIC_X COL_NAME_X+1+COL_NAME_WIDTH
#define COL_PATRONYMIC_WIDTH 17
#define COL_YOB_X COL_PATRONYMIC_X+1+COL_PATRONYMIC_WIDTH
#define COL_YOB_WIDTH 5
#define COL_GENDER_X COL_YOB_X+1+COL_YOB_WIDTH
#define COL_GENDER_WIDTH 3
#define COL_PROF_X COL_GENDER_X+1+COL_GENDER_WIDTH
#define COL_PROF_WIDTH 17
#define COL_EXP_X COL_PROF_X+1+COL_PROF_WIDTH
#define COL_EXP_WIDTH 3
#define COL_CLASS_X COL_EXP_X+1+COL_EXP_WIDTH
#define COL_CLASS_WIDTH 3
#define COL_DEPT_X COL_CLASS_X+1+COL_CLASS_WIDTH
#define COL_DEPT_WIDTH 3
#define COL_PLOT_X COL_DEPT_X+1+COL_DEPT_WIDTH
#define COL_PLOT_WIDTH 3
#define COL_SALARY_X COL_PLOT_X+1+COL_PLOT_WIDTH
#define COL_SALARY_WIDTH 8

Layout* mainLayout;
Component* scrollBar;
Component* labels[COLUMNS_COUNT] = {0};
Component** columns[COLUMNS_COUNT] = {0};

Component* idColLabel;
Component* colId[MAX_TABLE_SIZE];
Component* surnameColLabel;
Component* colSurname[MAX_TABLE_SIZE];
Component* nameColLabel;
Component* colName[MAX_TABLE_SIZE];
Component* patronymicColLabel;
Component* colPatronymic[MAX_TABLE_SIZE];
Component* yobColLabel;
Component* colYOB[MAX_TABLE_SIZE];
Component* genderColLabel;
Component* colGender[MAX_TABLE_SIZE];
Component* profColLabel;
Component* colProf[MAX_TABLE_SIZE];
Component* expColLabel;
Component* colExp[MAX_TABLE_SIZE];
Component* classColLabel;
Component* colClass[MAX_TABLE_SIZE];
Component* deptColLabel;
Component* colDept[MAX_TABLE_SIZE];
Component* plotColLabel;
Component* colPlot[MAX_TABLE_SIZE];
Component* salaryColLabel;
Component* colSalary[MAX_TABLE_SIZE];

bool running = true;
int currentTableSize = 0;
Array* currentData = NULL;
int currentPos = 0;

void OnColumnDirectionChange(Component* handle) {
    for (int i = 0; i < COLUMNS_COUNT; i++) {
        if (labels[i] != NULL && labels[i] != handle) {
            ColumnLabelSetDirection(labels[i], NONE);
        }
    }
    ColumnLabel* columnLabel = handle->spec;

```

```

        SortData(columnLabel->fieldId, columnLabel->activeDirection);
    }

    void createIdCol() {
        idCollabel = CreateColumnLabel(columnLabelStyle,
            COL_ID_X, TABLE_LABEL_Y, COL_ID_WIDTH, FIELD_ID, L"T/H");
        labels[FIELD_ID] = idCollabel;
        columns[FIELD_ID] = colId;
        ColumnLabel* columnLabel = idCollabel->spec;
        columnLabel->OnDirectionChange = OnColumnDirectionChange;
        LayoutAddComponent(mainLayout, idCollabel);
        for (int i = 0; i < MAX_TABLE_SIZE; i++) {
            colId[i] = CreateButton(i % 2 == 0 ? evenButtonStyle : oddButtonStyle,
                COL_ID_X, TABLE_BODY_START_Y + i, COL_ID_WIDTH, L"", EditEntry);
            colId[i]->OnKeyClick = OnIdButtonKeyClick;
            ((Button*) colId[i]->spec)->panel->OnMouseClicked = OnIdButtonMouseClicked;
        }
    }

    void createSurnameCol() {
        surnameCollabel = CreateColumnLabel(columnLabelStyle,
            COL_SURNAME_X, TABLE_LABEL_Y, COL_SURNAME_WIDTH, FIELD_SURNAME, L"Фамилия");
        labels[FIELD_SURNAME] = surnameCollabel;
        columns[FIELD_SURNAME] = colSurname;
        ColumnLabel* columnLabel = surnameCollabel->spec;
        columnLabel->OnDirectionChange = OnColumnDirectionChange;
        LayoutAddComponent(mainLayout, surnameCollabel);
        for (int i = 0; i < MAX_TABLE_SIZE; i++) {
            colSurname[i] = CreateEdit(i % 2 == 0 ? evenEditStyle : oddEditStyle,
                COL_SURNAME_X, TABLE_BODY_START_Y + i, COL_SURNAME_WIDTH - 1);
            colSurname[i]->tabFocusing = false;
            EditSetEnterAction(colSurname[i], ChangeSurname);
            LayoutAddComponent(mainLayout, colSurname[i]);
        }
    }

    void createNameCol() {
        nameCollabel = CreateColumnLabel(columnLabelStyle,
            COL_NAME_X, TABLE_LABEL_Y, COL_NAME_WIDTH, FIELD_NAME, L"Имя");
        labels[FIELD_NAME] = nameCollabel;
        columns[FIELD_NAME] = colName;
        ColumnLabel* columnLabel = nameCollabel->spec;
        columnLabel->OnDirectionChange = OnColumnDirectionChange;
        LayoutAddComponent(mainLayout, nameCollabel);
        for (int i = 0; i < MAX_TABLE_SIZE; i++) {
            colName[i] = CreateEdit(i % 2 == 0 ? evenEditStyle : oddEditStyle,
                COL_NAME_X, TABLE_BODY_START_Y + i, COL_NAME_WIDTH - 1);
            colName[i]->tabFocusing = false;
            EditSetEnterAction(colName[i], ChangeName);
            LayoutAddComponent(mainLayout, colName[i]);
        }
    }

    void createPatronymicCol() {
        patronymicCollabel = CreateColumnLabel(columnLabelStyle,
            COL_PATRONYMIC_X, TABLE_LABEL_Y, COL_PATRONYMIC_WIDTH, FIELD_PATRONYMIC, L"Отчество");
        labels[FIELD_PATRONYMIC] = patronymicCollabel;
        columns[FIELD_PATRONYMIC] = colPatronymic;
        ColumnLabel* columnLabel = patronymicCollabel->spec;
        columnLabel->OnDirectionChange = OnColumnDirectionChange;
        LayoutAddComponent(mainLayout, patronymicCollabel);
        for (int i = 0; i < MAX_TABLE_SIZE; i++) {
            colPatronymic[i] = CreateEdit(i % 2 == 0 ? evenEditStyle : oddEditStyle,
                COL_PATRONYMIC_X, TABLE_BODY_START_Y + i, COL_PATRONYMIC_WIDTH - 1);
            colPatronymic[i]->tabFocusing = false;
            EditSetEnterAction(colPatronymic[i], ChangePatronymic);
            LayoutAddComponent(mainLayout, colPatronymic[i]);
        }
    }

    void createYOBCol() {
        yobCollabel = CreateColumnLabel(columnLabelStyle,
            COL_YOB_X, TABLE_LABEL_Y, COL_YOB_WIDTH, FIELD_YOB, L"Г.р.");
        labels[FIELD_YOB] = yobCollabel;
        columns[FIELD_YOB] = colYOB;
        ColumnLabel* columnLabel = yobCollabel->spec;
    }

```

```

columnLabel->OnDirectionChange = OnColumnDirectionChange;
LayoutAddComponent(mainLayout, yobColLabel);
for (int i = 0; i < MAX_TABLE_SIZE; i++) {
    colYOB[i] = CreateEdit(i % 2 == 0 ? evenEditStyle : oddEditStyle,
        COL_YOB_X, TABLE_BODY_START_Y + i, COL_YOB_WIDTH - 1);
    colYOB[i]->tabFocusing = false;
    EditSetFilter(colYOB[i], PositiveNumberFilter);
    EditSetEnterAction(colYOB[i], ChangeYOB);
    LayoutAddComponent(mainLayout, colYOB[i]);
}
}

void createGenderCol() {
    genderColLabel = CreateColumnLabel(columnLabelStyle,
        COL_GENDER_X, TABLE_LABEL_Y, COL_GENDER_WIDTH, FIELD_GENDER, L"Пол");
    labels[FIELD_GENDER] = genderColLabel;
    columns[FIELD_GENDER] = colGender;
    ColumnLabel* columnLabel = genderColLabel->spec;
    columnLabel->OnDirectionChange = OnColumnDirectionChange;
    LayoutAddComponent(mainLayout, genderColLabel);
    for (int i = 0; i < MAX_TABLE_SIZE; i++) {
        colGender[i] = CreateSelect(i % 2 == 0 ? evenSelectStyle : oddSelectStyle,
            COL_GENDER_X, TABLE_BODY_START_Y + i, COL_GENDER_WIDTH, 2, L"М", L"Ж");
        colGender[i]->tabFocusing = false;
        SelectSetEnterAction(colGender[i], ChangeGender);
        LayoutAddComponent(mainLayout, colGender[i]);
    }
}

void createProfCol() {
    profColLabel = CreateColumnLabel(columnLabelStyle,
        COL_PROF_X, TABLE_LABEL_Y, COL_PROF_WIDTH, FIELD_PROFESSION, L"Профессия");
    labels[FIELD_PROFESSION] = profColLabel;
    columns[FIELD_PROFESSION] = colProf;
    ColumnLabel* columnLabel = profColLabel->spec;
    columnLabel->OnDirectionChange = OnColumnDirectionChange;
    LayoutAddComponent(mainLayout, profColLabel);
    for (int i = 0; i < MAX_TABLE_SIZE; i++) {
        colProf[i] = CreateEdit(i % 2 == 0 ? evenEditStyle : oddEditStyle,
            COL_PROF_X, TABLE_BODY_START_Y + i, COL_PROF_WIDTH - 1);
        colProf[i]->tabFocusing = false;
        EditSetEnterAction(colProf[i], ChangeProfession);
        LayoutAddComponent(mainLayout, colProf[i]);
    }
}

void createExpCol() {
    expColLabel = CreateColumnLabel(columnLabelStyle,
        COL_EXP_X, TABLE_LABEL_Y, COL_EXP_WIDTH, FIELD_EXPERIENCE, L"Оп.");
    labels[FIELD_EXPERIENCE] = expColLabel;
    columns[FIELD_EXPERIENCE] = colExp;
    ColumnLabel* columnLabel = expColLabel->spec;
    columnLabel->OnDirectionChange = OnColumnDirectionChange;
    LayoutAddComponent(mainLayout, expColLabel);
    for (int i = 0; i < MAX_TABLE_SIZE; i++) {
        colExp[i] = CreateEdit(i % 2 == 0 ? evenEditStyle : oddEditStyle,
            COL_EXP_X, TABLE_BODY_START_Y + i, COL_EXP_WIDTH - 1);
        colExp[i]->tabFocusing = false;
        EditSetFilter(colExp[i], PositiveNumberFilter);
        EditSetEnterAction(colExp[i], ChangeExperience);
        LayoutAddComponent(mainLayout, colExp[i]);
    }
}

void createClassCol() {
    classColLabel = CreateColumnLabel(columnLabelStyle,
        COL_CLASS_X, TABLE_LABEL_Y, COL_CLASS_WIDTH, FIELD_CLASS, L"Р-д");
    labels[FIELD_CLASS] = classColLabel;
    columns[FIELD_CLASS] = colClass;
    ColumnLabel* columnLabel = classColLabel->spec;
    columnLabel->OnDirectionChange = OnColumnDirectionChange;
    LayoutAddComponent(mainLayout, classColLabel);
    for (int i = 0; i < MAX_TABLE_SIZE; i++) {
        colClass[i] = CreateSelect(i % 2 == 0 ? evenSelectStyle : oddSelectStyle,
            COL_CLASS_X, TABLE_BODY_START_Y + i, COL_CLASS_WIDTH, 3, L"1", L"2", L"3");
        colClass[i]->tabFocusing = false;
    }
}

```

```

        SelectSetEnterAction(colClass[i], ChangeClass);
        LayoutAddComponent(mainLayout, colClass[i]);
    }
}

void createDeptCol() {
    deptCollabel = CreateColumnLabel(columnLabelStyle,
        COL_DEPT_X, TABLE_LABEL_Y, COL_DEPT_WIDTH, FIELD_DEPARTMENT, L"Цех");
    labels[FIELD_DEPARTMENT] = deptCollabel;
    columns[FIELD_DEPARTMENT] = colDept;
    ColumnLabel* columnLabel = deptCollabel->spec;
    columnLabel->OnDirectionChange = OnColumnDirectionChange;
    LayoutAddComponent(mainLayout, deptCollabel);
    for (int i = 0; i < MAX_TABLE_SIZE; i++) {
        colDept[i] = CreateEdit(i % 2 == 0 ? evenEditStyle : oddEditStyle,
            COL_DEPT_X, TABLE_BODY_START_Y + i, COL_DEPT_WIDTH - 1);
        colDept[i]->tabFocusing = false;
        EditSetFilter(colDept[i], PositiveNumberFilter);
        EditSetEnterAction(colDept[i], ChangeDepartment);
        LayoutAddComponent(mainLayout, colDept[i]);
    }
}

void createPlotCol() {
    plotCollabel = CreateColumnLabel(columnLabelStyle,
        COL_PLOT_X, TABLE_LABEL_Y, COL_PLOT_WIDTH, FIELD_PLOT, L"Уч.");
    labels[FIELD_PLOT] = plotCollabel;
    columns[FIELD_PLOT] = colPlot;
    ColumnLabel* columnLabel = plotCollabel->spec;
    columnLabel->OnDirectionChange = OnColumnDirectionChange;
    LayoutAddComponent(mainLayout, plotCollabel);
    for (int i = 0; i < MAX_TABLE_SIZE; i++) {
        colPlot[i] = CreateEdit(i % 2 == 0 ? evenEditStyle : oddEditStyle,
            COL_PLOT_X, TABLE_BODY_START_Y + i, COL_PLOT_WIDTH - 1);
        colPlot[i]->tabFocusing = false;
        EditSetFilter(colPlot[i], PositiveNumberFilter);
        EditSetEnterAction(colPlot[i], ChangePlot);
        LayoutAddComponent(mainLayout, colPlot[i]);
    }
}

void createSalaryCol() {
    salaryCollabel = CreateColumnLabel(columnLabelStyle,
        COL_SALARY_X, TABLE_LABEL_Y, COL_SALARY_WIDTH, FIELD_SALARY, L"Зарплата");
    labels[FIELD_SALARY] = salaryCollabel;
    columns[FIELD_SALARY] = colSalary;
    ColumnLabel* columnLabel = salaryCollabel->spec;
    columnLabel->OnDirectionChange = OnColumnDirectionChange;
    LayoutAddComponent(mainLayout, salaryCollabel);
    for (int i = 0; i < MAX_TABLE_SIZE; i++) {
        colSalary[i] = CreateEdit(i % 2 == 0 ? evenEditStyle : oddEditStyle,
            COL_SALARY_X, TABLE_BODY_START_Y + i, COL_SALARY_WIDTH - 1);
        colSalary[i]->tabFocusing = false;
        EditSetEnterAction(colSalary[i], ChangeSalary);
        LayoutAddComponent(mainLayout, colSalary[i]);
    }
}

void hideLines(int begin) {
    for (int l = begin; l < MAX_TABLE_SIZE; l++) {
        for (int c = 0; c < COLUMNS_COUNT; c++) {
            HideComponent(columns[c][l]);
        }
    }
}

void showLines(int begin, int count) {
    for (int l = begin; l < begin + count; l++) {
        for (int c = 0; c < COLUMNS_COUNT; c++) {
            ShowComponent(columns[c][l]);
        }
    }
}

void enableLabels(bool enabled) {
    for (int c = 0; c < COLUMNS_COUNT; c++) {

```

```

        ColumnLabelSetEnabled(labels[c], enabled);
    }
}

void drawTableHeader(bool disabled) {
    WINDOW* w = mainLayout->window;
    mvwchgat(w, 0, 0, WIDTH, 0, (short) menuStyle->defaultLabel, NULL);
    wattrset(w, COLOR_PAIR(disabled ? transitionMenuTableDisabled : transitionMenuTable));
    mvwhline(w, TABLE_LABEL_Y - 1, 0, ACS_BBLOCK, WIDTH);
    wattrset(w, COLOR_PAIR(disabled ? transitionMenuTableDisabled : transitionMenuTable));
    mvwhline(w, TABLE_LABEL_Y + 1, 0, ACS_BLOCK, WIDTH);
    wattrset(w, COLOR_PAIR(disabled ? tableDisabledColor : tableEvenColor));
    mvwaddch(w, TABLE_LABEL_Y, WIDTH - 2, ACS_RBLOCK);
    mvwaddch(w, TABLE_LABEL_Y, WIDTH - 1, ACS_BLOCK);
    for (int c = 0; c < COLUMNS_COUNT; c++) {
        int x = getbegx(((ColumnLabel*) labels[c]->spec)->panel->window) - 1;
        mvwaddch(w, TABLE_LABEL_Y, x, ACS_LBLOCK);
    }
}

void drawTableBody() {
    WINDOW* w = mainLayout->window;
    for (int l = 0; l < MAX_TABLE_SIZE; l++) {
        if (l < currentTableSize) {
            wattrset(w, COLOR_PAIR(l % 2 == 0 ? tableEvenColor : tableOddColor));
        } else {
            wattrset(w, COLOR_PAIR(tableEvenColor));
        }
        mvwaddch(w, TABLE_BODY_START_Y + 1, WIDTH - 2, ACS_RBLOCK);
        for (int c = 0; c < COLUMNS_COUNT; c++) {
            int x = getbegx(((ColumnLabel*) labels[c]->spec)->panel->window) - 1;
            mvwaddch(w, TABLE_BODY_START_Y + 1, x, ACS_LBLOCK);
        }
    }
}

void drawStarter() {
    WINDOW* w = mainLayout->window;
    wattrset(w, 0);
    int x = ((int) (WIDTH - wcslen(logo[0]))) / 2;
    for (int i = 0; i < LOGO_SIZE; i++) {
        mvwaddwstr(w, LOGO_Y + i, x, logo[i]);
    }
    mvwaddstr(w, VERSION_Y, (WIDTH - 17) / 2, "Manufactury v" MANUFACTURY_VERSION);
    mvwaddstr(w, HELP_Y, (WIDTH - 69) / 2, "Для начала работы откройте файл(Ctrl+O) или создайте  
новый(Ctrl+N)...");
}

void drawTableFooter(bool disabled) {
    WINDOW* w = mainLayout->window;
    wattrset(w, COLOR_PAIR(disabled ? transitionMenuTableDisabled : transitionMenuTable));
    mvwhline(w, TABLE_BODY_START_Y + MAX_TABLE_SIZE, 0, ACS_UBLOCK, WIDTH);
}

void showLine(int l, Employee* employee) {
    wchar_t id[COL_ID_WIDTH];
    swprintf(id, COL_ID_WIDTH, L"%d", employee->id);
    ButtonSetText(colId[l], id);
    EditSetValue(colSurname[l], employee->surname);
    EditSetValue(colName[l], employee->name);
    EditSetValue(colPatronymic[l], employee->patronymic);
    wchar_t yob[COL_YOB_WIDTH];
    swprintf(yob, COL_YOB_WIDTH, L"%d", employee->yob);
    EditSetValue(colYOB[l], yob);
    SelectSetValue(colGender[l], employee->gender ? 0 : 1);
    EditSetValue(colProf[l], employee->profession);
    wchar_t exp[COL_EXP_WIDTH];
    swprintf(exp, COL_EXP_WIDTH, L"%d", employee->experience);
    EditSetValue(colExp[l], exp);
    SelectSetValue(colClass[l], employee->class - 1);
    wchar_t dept[COL_DEPT_WIDTH];
    swprintf(dept, COL_DEPT_WIDTH, L"%d", employee->department);
    EditSetValue(colDept[l], dept);
    wchar_t plot[COL_PLOT_WIDTH];
    swprintf(plot, COL_PLOT_WIDTH, L"%d", employee->plot);
    EditSetValue(colPlot[l], plot);
}

```



```

        wchar_t salary[COL_SALARY_WIDTH];
        swprintf(salary, COL_SALARY_WIDTH, L"%d", employee->salary);
        EditSetValue(colSalary[1], salary);
    }

    void showData(int pos) {
        currentPos = pos;
        int newSize = min((int) (array_size(currentData) - currentPos), MAX_TABLE_SIZE);
        if (newSize != currentTableSize) {
            if (newSize > currentTableSize) {
                showLines(currentTableSize, newSize - currentTableSize);
            } else {
                hideLines(newSize);
            }
            currentTableSize = newSize;
            drawTableBody();
        }
        for (int l = 0; l < currentTableSize; l++) {
            Employee* employee;
            array_get_at(currentData, (size_t) pos + l, (void**) &employee);
            for (int c = 0; c < COLUMNS_COUNT; c++) {
                columns[c][l]->custom = employee;
            }
            showLine(l, employee);
        }
        ScrollBarSetNumber(scrollBar, pos);
    }

    void InitMainView(void) {
        resize_term(HEIGHT, WIDTH);

        mainLayout = CreateLayout(0, 0, WIDTH, HEIGHT);
        wbkgd(mainLayout->window, COLOR_PAIR(mainBackground));

        Component* menu1 = CreateMenu(menuStyle, 0, 0, L"Файл", 4,
            L"Новый", CreateHotKey('N', KEY_CTRL), FileNew,
            L"Открыть", CreateHotKey('O', KEY_CTRL), FileOpen,
            L"Сохранить", CreateHotKey('S', KEY_CTRL), FileSave,
            L"Сохранить как...", CreateHotKey('S', KEY_CTRL | KEY_SHIFT),
FileSaveAs);
        Component* menu2 = CreateMenu(menuStyle, 6, 0, L"Правка", 5,
            L"Найти", CreateHotKey('F', KEY_CTRL), EditFind,
            L"Отменить поиск", CreateHotKey('F', KEY_ALT), EditCancelFind,
            L"Добавить", CreateHotKey('I', KEY_CTRL), EditAdd,
            L"Удалить", CreateHotKey('D', KEY_CTRL), EditDelete,
            L"Изменить", CreateHotKey('E', KEY_CTRL), EditChange);
        Component* menu3 = CreateMenu(menuStyle, 14, 0, L"Инструменты", 2,
            L"Экспорт в .CSV", CreateHotKey('E', KEY_CTRL | KEY_ALT), ToolsEx-
portCSV,
            L"Создать отчет", CreateHotKey('R', KEY_CTRL), ToolsCreateR-
eport);

        LayoutAddComponent(mainLayout, menu1);
        LayoutAddComponent(mainLayout, menu2);
        LayoutAddComponent(mainLayout, menu3);

        createIdCol();
        createSurnameCol();
        createNameCol();
        createPatronymicCol();
        createYOBCol();
        createGenderCol();
        createProfCol();
        createExpCol();
        createClassCol();
        createDeptCol();
        createPlotCol();
        createSalaryCol();
        for (int l = 0; l < MAX_TABLE_SIZE; l++) {
            LayoutAddComponent(mainLayout, colId[l]);
        }

        scrollBar = CreateScrollBar(scrollBarStyle, WIDTH - 1, TABLE_BODY_START_Y, MAX_TABLE_SIZE, main-
Layout);
        LayoutAddComponent(mainLayout, scrollBar);

        mainLayout->OnScrollDown = TableOnScrollDown;
    }

```

```

mainLayout->OnScrollUp = TableOnScrollUp;

InitLayouts(mainLayout);

update_panels();
douupdate();
}

void StartControl(void) {
    MEVENT event;
    int input;
    while (running) {
        input = getch();
        if (input != ERR) {
            unsigned long modifiers = PDC_get_key_modifiers();
            if (input == KEY_RESIZE) {
                resize_term(0, 0);
                if (LINES != HEIGHT || COLS != WIDTH) {
                    ShowMessage(L"Изменение размеров окна не поддерживается");
                    running = false;
                    return;
                }
                resize_term(HEIGHT, WIDTH);
                PANEL* panel = panel_below(NULL);
                while (panel != NULL) {
                    touchwin(panel_window(panel));
                    panel = panel_below(panel);
                }
            } else if (input == KEY_MOUSE) {
                nc_getmouse(&event);
                LayoutHandleMouseEvent(event);
            } else {
                if (!HandleHotKeyEvent(input, modifiers)) {
                    LayoutHandleKeyboardEvent(input, modifiers);
                }
            }
            update_panels();
            douupdate();
        }
    }
}

void SetPos(int pos) {
    showData(pos);
}

void SetData(Array* data) {
    currentData = data;
    ScrollBarSetCount(scrollBar, (int) array_size(currentData));
    showData(currentPos);
}

void ShowStarter(void) {
    enableLabels(false);
    HideComponent(scrollBar);
    hideLines(0);
    currentTableSize = 0;
    wclear(mainLayout->window);
    drawTableHeader(true);
    drawStarter();
    drawTableFooter(true);
    update_panels();
    douupdate();
}

void ShowTable(void) {
    enableLabels(true);
    ShowComponent(scrollBar);
    hideLines(0);
    currentTableSize = 0;
    wclear(mainLayout->window);
    drawTableHeader(false);
    drawTableBody();
    drawTableFooter(false);
    update_panels();
    douupdate();
}

```

```

}

void ResetActiveColumnLabel(void) {
    for (int i = 0; i < COLUMNS_COUNT; i++) {
        ColumnLabelSetDirection(labels[i], NONE);
    }
}

void ShowFileReadError(void) {
    ShowMessageDialog(L"Ошибка чтения файла", NULL);
}

void ShowFileWriteError(void) {
    ShowMessageDialog(L"Ошибка записи файла", NULL);
}

void ShowFilterError(void) {
    ShowMessageDialog(L"Записи не найдены", NULL);
}

void ShowEntryChanges(Employee* e) {
    for (int l = 0; l < currentTableSize; l++) {
        Employee* employee;
        array_get_at(currentData, (size_t) currentPos + 1, (void**) &employee);
        if (employee == e) {
            showLine(l, employee);
            break;
        }
    }
}

```

Приложение Б. Алгоритмы функций

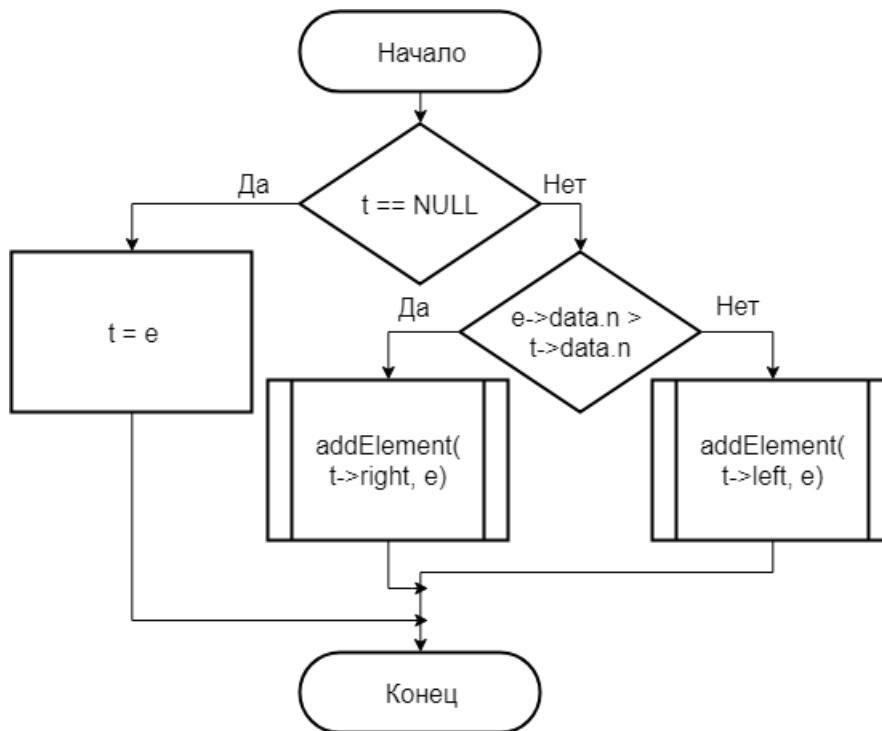


Рисунок Б.1 – Алгоритм функции добавления элемента в структуру

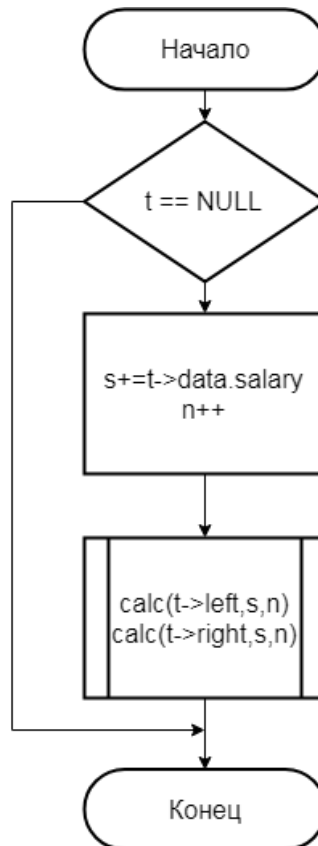


Рисунок Б.2 – Алгоритм расчета результатов

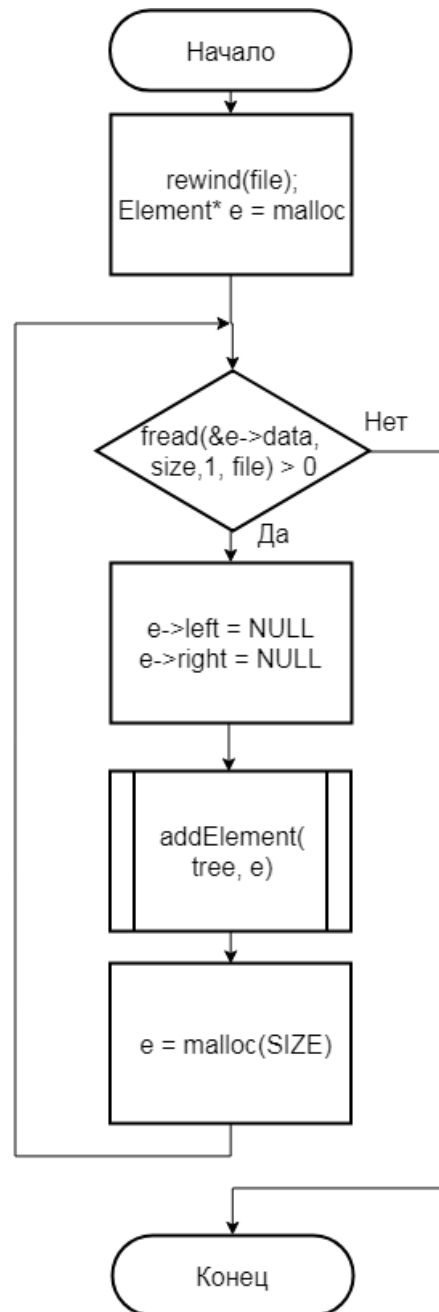


Рисунок Б.3 – Алгоритм загрузки данных из бинарного файла

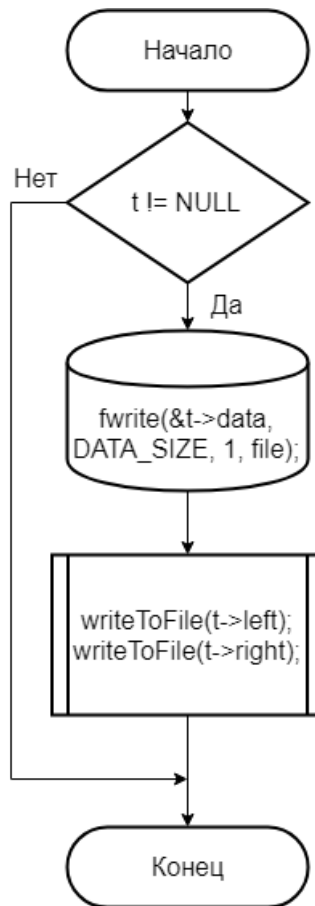


Рисунок Б.4 – Алгоритм сохранения данных в бинарный файл

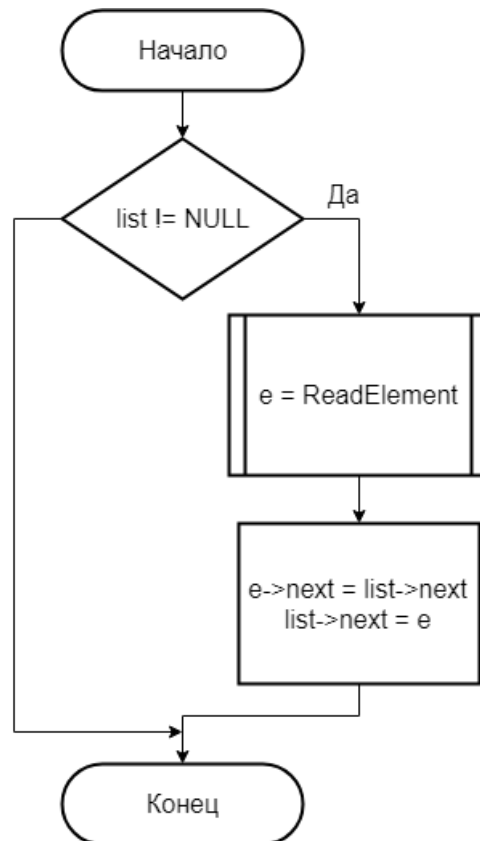


Рисунок Б.5 – Алгоритм вставки в элемента на 2-ую позицию

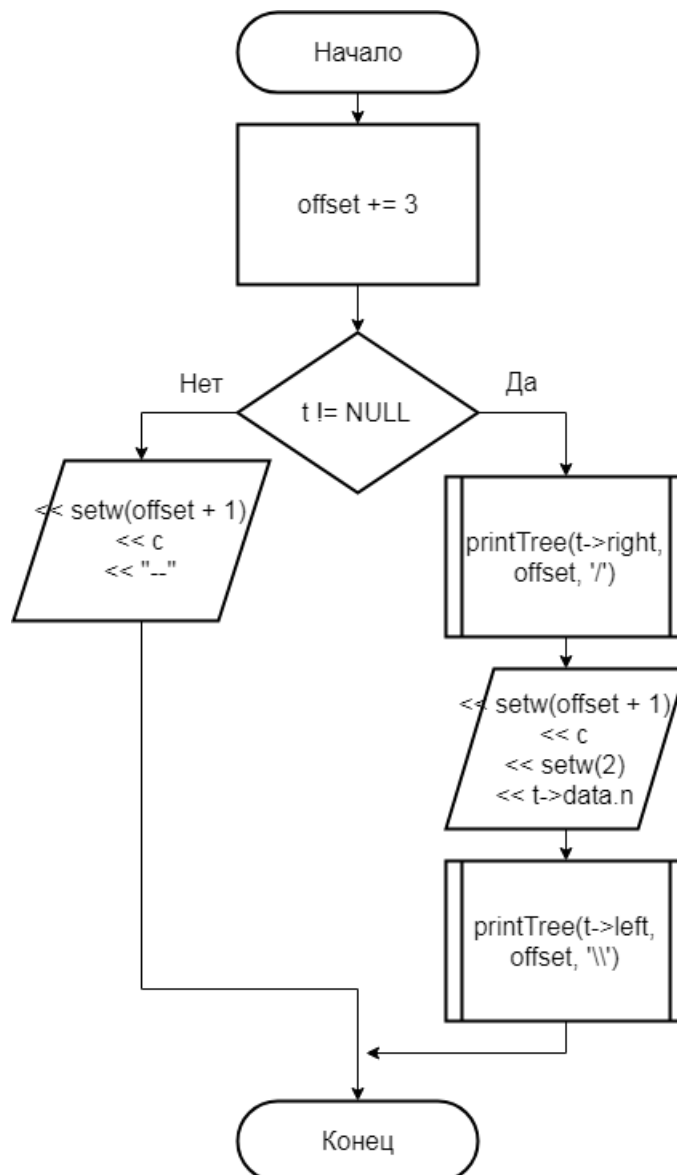


Рисунок Б.6 – Алгоритм вывода данных структуры в консоль

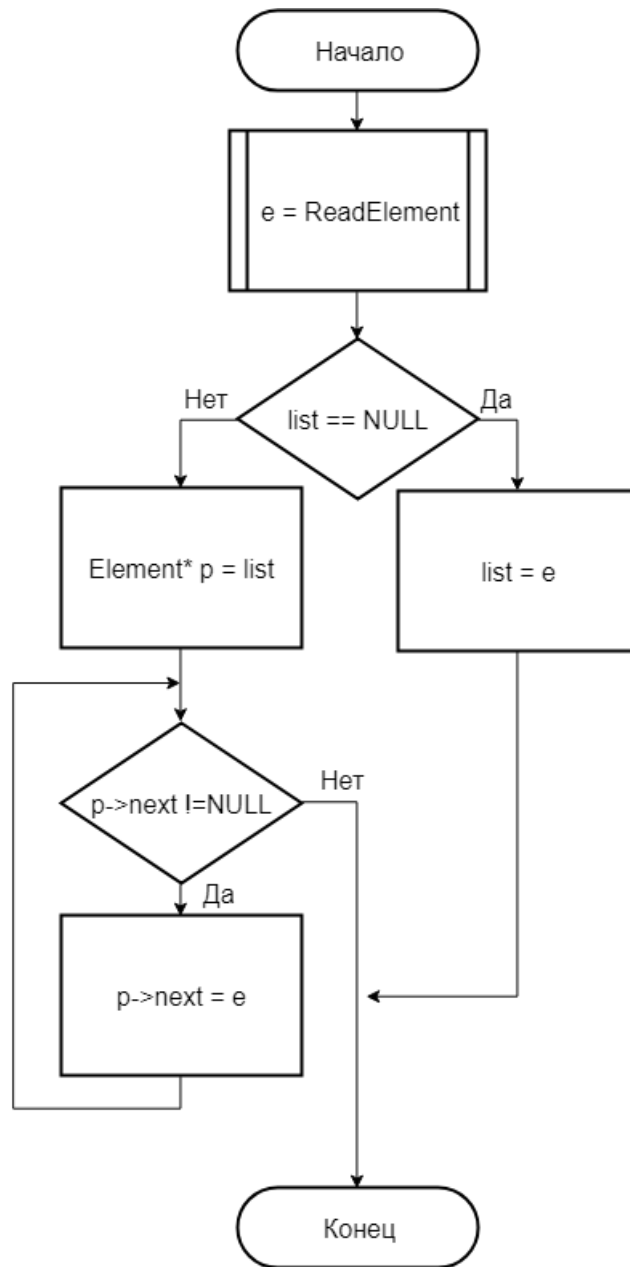


Рисунок Б.7 – Алгоритм добавления элемента в конец списка

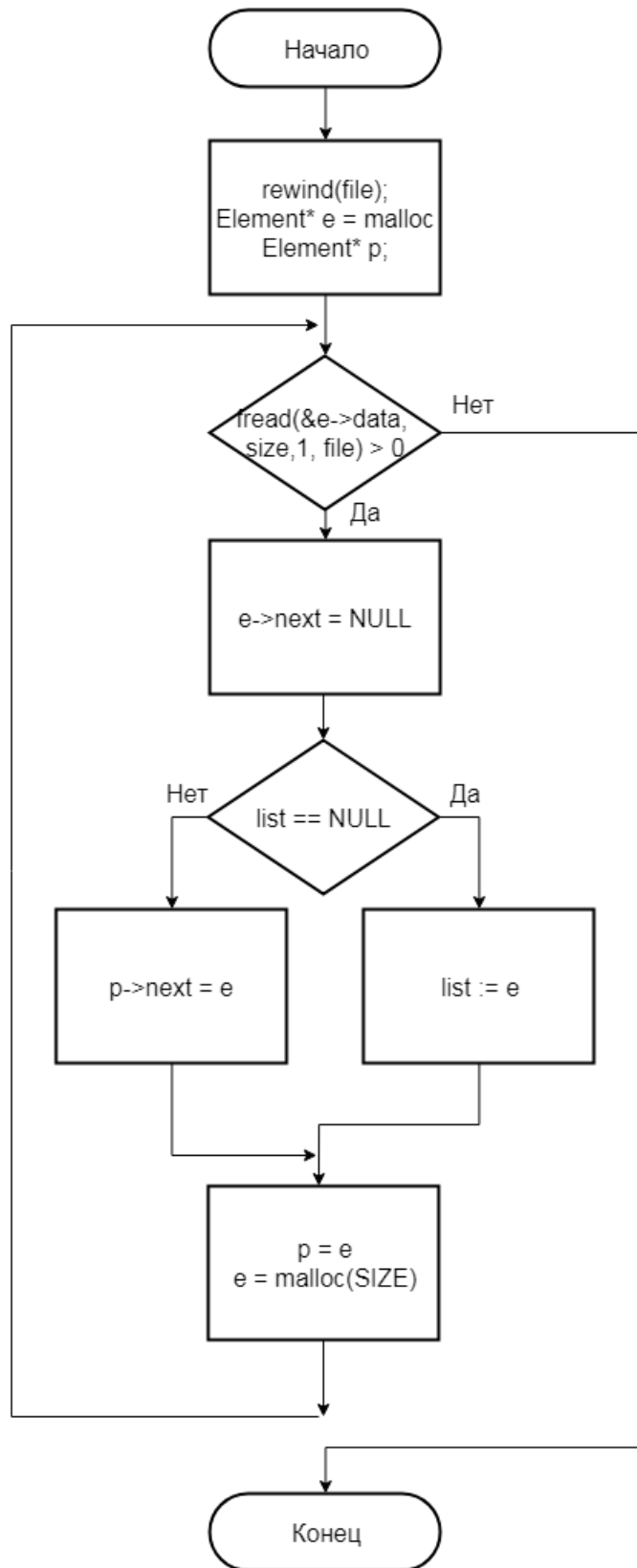


Рисунок Б.8 – Алгоритм загрузки списка из файла