

АННОТАЦИЯ

Целью курсового проектирования является систематизация, закрепление и углубление знаний в области основ программирования и совершенствование практических навыков разработки программ на языках C/C++ с использованием методологии структурного программирования. Для достижения цели на разных этапах курсового проектирования должны быть решены следующие задачи:

- выбор варианта задания и детализация поставки задачи; - определение требований к функциям, выполняемых разрабатываемой программой;
- выбор типов и проектирование структур данных, определяющих способы представления, хранения и преобразования входных, выходных и промежуточных данных;
- разработка модульной структуры программы, определение функций модулей и способов их взаимодействия;
- написание текста (кодирование) программных модулей на алгоритмическом языке;
- разработка тестовых примеров;
- тестирование и отладка программы;
- разработка программных документов в соответствии с действующими стандартами.

Выполнение курсовой работы осуществляется по индивидуальному заданию, в соответствии с которым необходимо разработать программу для хранения и обработки информации, представленной в форме таблицы. Подобные программы являются важной составной частью различных автоматизированных информационных систем. Приобретение студентами

практического опыта в разработке таких программ обеспечит хороший фундамент для освоения последующих дисциплин, связанных с разработкой информационных систем.

В рамках настоящего курсового проектирования при разработке программы используется методология структурного программирования, которая базируется на методах нисходящего проектирования и модульного программирования, структурного кодирования программ [4, 8]. Соблюдение этой методологии позволяет сократить время разработки программ и повысить их качество.

При выполнении курсовой работы необходимо учитывать, что разработка программных систем – это итеративный процесс. При этом основные фазы создания программы – проектирование, программирование и тестирование – нельзя строго разделить и зачастую они выполняются параллельно либо со значительным перекрытием во времени.

ВВЕДЕНИЕ

В рамках данного курсового проектирования, представляем программу “Учет и управление данными о материальных ценностях”, разработанную на основе технического задания в организации Севастопольский государственный университет, утвержденного 01.09.2023. Эта программа представляет собой инструмент для эффективного учета и управления информацией о материальных ценностях.

Управление данными о материальных ценностях имеет важное значение для организаций. Это позволяет хранить, обновлять и анализировать информацию о материальных ценностях, что в свою очередь способствует оптимизации процессов управления ресурсами и принятию обоснованных решений в области управления материальными ценностями.

Целью данного курсового проекта является разработка программы "Учет и управление данными о материальных ценностях", предоставляющей средства для эффективного учета информации о материальных ценностях в организации, а также возможность управления этими данными. Программа поможет улучшить организацию и управление материальными ценностями.

1. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ ПРОГРАММЫ

Программа "Учет и управление материальными ценностями" предназначена для систематизации и эффективного управления информацией о материальных ценностях в организации. Ее основная цель состоит в создании удобного и функционального инструмента, который поможет облегчить процессы учета материальных ценностей и оптимизировать операции, связанные с данными о них.

Программа может успешно применяться в следующих областях:

1. Учет материальных ценностей: Ведение информации о материальных ценностях, включая их характеристики, стоимость, количество и местонахождение.

2. Управление запасами: Оптимизация и контроль за запасами материальных ценностей для предотвращения избыточных закупок или нехватки материалов.

3. Планирование закупок и расходов: Анализ данных о материальных ценностях для точного планирования закупок и оптимизации расходов.

4. Отчетность и аналитика: Генерация отчетов и аналитических данных для принятия информированных решений в управлении материальными ценностями.

Цель программы заключается в автоматизации учета и управления данными о материальных ценностях для повышения эффективности и точности управления материальными ресурсами в организации.

2. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ ПРОГРАММЫ

2.1 Постановка задачи на разработку программы

Вариант задания: Даны сведения за квартал о материальных ценностях в стоимостном выражении по филиалам завода. Структура записи: номер завода, номер филиала, фамилия ответственного за материальные ценности, наличие материальных ценностей на начало периода, получено материальных ценностей на сумму, выбыло на сумму. Получить ведомость движения материальных ценностей за отчетный период, содержащую: стоимость материальных ценностей по каждому филиалу на конец отчетного периода, по всему заводу на конец периода, а также итоговые цифры по каждому виду (на начало периода, получено, выбыло, на конец периода) и по всему заводу в целом. Проконтролировать двумя путями (по строке и столбцу) получение итоговой цифры: наличие материальных ценностей на конец отчетного периода в стоимостном выражении по заводу в целом

Для представления таблицы в коде программы необходимо использовать списки (однонаправленные или двунаправленные) или бинарные деревья. Запрещается использовать классы или готовые контейнеры.

Разрабатываемая программа должна использовать меню-ориентированный интерфейс, обеспечивающий выполнение следующего минимального состава действий.

Программа должна обеспечивать следующие функции:

1. Начальное создание таблицы. При необходимости создания новой таблицы исходные данные считываются из текстового файла. Имя файла должен задавать пользователь.

2. Просмотр таблицы. При этом необходимо предусмотреть возможность скроллинга.
3. Добавление новой записи в таблицу.
4. Удаление записи. Удаляемый элемент выбирается по одному из полей таблицы (ключевому). Ключевое поле выбирает студент.
5. Корректировка записи в таблице. Корректируемую запись выбирают по одному из полей таблицы (ключевому).
6. Сортировка таблицы. Сортировка производится по одному из полей таблицы (ключевому). Метод сортировки выбирает студент.
7. Поиск записи в таблице (по не ключевому полю).
8. Сохранение таблицы в файле. Имя файла должен вводить пользователь. Сохранять таблицу следует в текстовый файл и в типизированный.
9. Чтение данных из файла. Имя файла должен вводить пользователь. При чтении данных необходимо создавать новый список.
10. Обработка таблицы и просмотр результатов обработки. Результат обработки необходимо вывести на экран и в текстовый файл. Имя файла вводит пользователь.
11. Выход — завершение работы программы.

2.2 Описание и обоснование выбора метода организации входных и выходных данных

В качестве структуры данных для программы был выбран односвязный список. Реализация динамической структуры в виде односвязного списка имеет следующие преимущества:

- Экономия памяти при добавлении элементов: Односвязный список выделяет дополнительную память только для каждого нового элемента. В отличие от динамического массива, где требуется выделение новой области памяти для всех элементов, копирование старых данных в новую область и только потом освобождение памяти, выделенной под старую область. При увеличении количества элементов в структуре данных, это преимущество односвязного списка становится более заметным.

- Меньшее количество дополнительных операций при манипулировании данными: Односвязный список требует меньше операций при добавлении нового элемента или удалении существующего. Это обусловлено отсутствием необходимости перемещать большие блоки данных, как это требуется в случае динамического массива.

В качестве входных и выходных данных используется текстовый файл, поля которого задаёт пользователь. Данные легко читаются и изменяются в файле без необходимости кодирования. Это обеспечивает удобство в работе с информацией, а также улучшает ее портируемость и доступность для редактирования.

2.3 Обоснование выбора языка и среды программирования

В качестве языка программирования был выбран язык C с элементами C++ в среде разработки Embarcadero Dev-C++ 6.3.

Компилируемый статически типизированный язык программирования C был разработан сотрудником Bell Labs, Деннисом Ритчи, в период с 1969 по 1973 годы. Изначально созданный для реализации операционной системы UNIX, он был адаптирован для работы на различных платформах. Дизайн языка был ориентирован на соответствие машинным инструкциям, что

обеспечило его широкое использование в проектах, где требовался язык программирования, близкий к ассемблеру.

Язык С применяется в различных областях, включая операционные системы и прикладное программное обеспечение для широкого спектра устройств: от суперкомпьютеров до встраиваемых систем. Эта универсальность и мощь делают его популярным выбором для разработчиков, работающих в сферах, где требуется эффективное управление ресурсами и высокая производительность.

С является одним из наиболее влиятельных языков программирования и оказал значительное влияние на развитие компьютерной индустрии. Его особенности, такие как эффективность, портируемость и близость к аппаратному уровню, продолжают делать его популярным средством разработки программного обеспечения в различных областях информационных технологий.

К преимуществам языка можно отнести:

- Универсальность (используется практически во всех существующих ЭВМ)
- Малое потребление ОЗУ
- Малый размер исполняемого файла
- Быстрота компилирования и выполнения программ
- Гибкость написания сложных многоструктурных программ
- Высокая структурированность

Но главным критерием выбора данного языка программирования стало изучение данного языка программирования и освоение его на практике в течение первого курса обучения и именно на нём в соответствии с техническим заданием, должна быть написана программа.

2.4 Описание алгоритмов функционирования программы

Для выполнения КП были разработаны следующие функции:

- `MaterialRecord* createRecord()` — Функция для создания новой записи;
- `void addRecord()` — Функция для добавления записи в таблицу;
- `void deleteRecord()` — Функция для удаления записи из таблицы по ключевому полю;
- `void editRecord()` — Функция для корректировки записи в таблице по ключевому полю;
- `void displayRecords()` — Просмотр таблицы с возможностью скроллинга;
- `void sortRecords()` — Функция для сортировки таблицы по нескольким полям;
- `void searchByLastName()` — Функция для поиска записи в таблице по не ключевому полю (Фамилия);
- `void saveToTextFile()` — Функция для сохранения таблицы в текстовом файле;
- `void saveToBinaryFile()` — Функция для сохранения таблицы в бинарном файле;
- `void loadFromTextFile()` — Создание таблицы из текстового файла;
- `bool compareByColumnUp()` — Вспомогательная функция для сортировки по возрастанию;
- `bool compareByColumnDown()` — Вспомогательная функция для сортировки по убыванию;
- `void swap()` — Вспомогательная функция обмена элементов для функции `sortRecords`;

- void releaseMemory() — Функция очистки памяти
- void exitProgram() — Функция выхода из программы;
- void generateReport() — Функция выбора обработки таблицы;
- void printReport() — Функция вывода полного отчёта на экран;
- void printReportToFile() — Функция вывода полного отчёта в файл;
- void calculateEndPeriodValueByBranch() — Функция подсчёта и вывода итоговых цифр по каждому филиалу;
- void calculateEndPeriodValueByFactory() — Функция подсчёта и вывода итоговых цифр по каждому заводу;
- void calculateTotals() — Функция подсчёта и вывода итоговых цифр по всем заводам.

На рисунках ниже представлены структурные схемы некоторых алгоритмов (функций) программы.

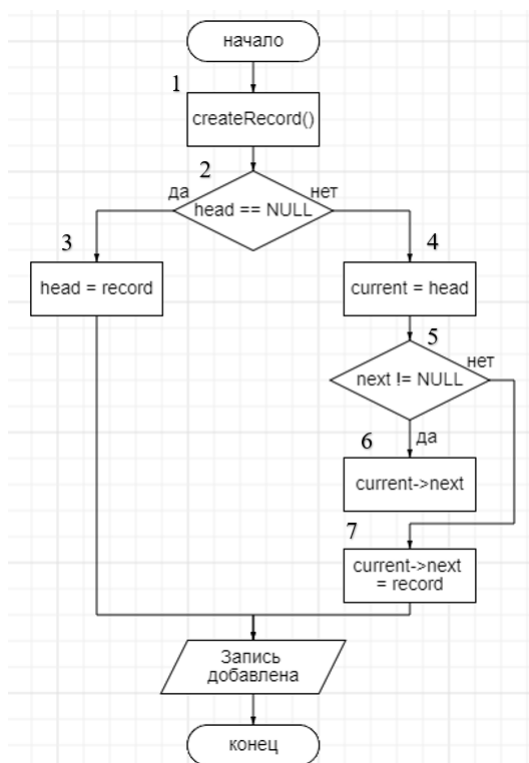


Рисунок 2.1 – Алгоритм функции addRecord

На рисунке 2.1 представлена функция добавление записи в таблицу.

Поблочное описание функции:

Блок 1 – указатель record на структуру Record и присваивает ему значение, возвращаемое функцией createRecord().

Блок 2 – проверяет, если переменная head, являющаяся указателем на первую запись в таблице, равна NULL.

Блок 3 – Если head равен NULL, то эта строка присваивает ему значение указателя record.

Блок 4 – создает указатель current на структуру Record и присваивает ему значение head.

Блок 5 – запускает цикл while, который выполняется, пока у текущей записи temp есть следующая запись.

Блок 6 – внутри цикла while эта строка переходит к следующей записи в связном списке, присваивая current значение temp->next.

Блок 7 – После выхода из цикла while, эта строка устанавливает указатель next последней записи в связном списке (temp) на новую запись record. Таким образом, новая запись добавляется в конец списка.



Рисунок 2.2 – Алгоритм функции createRecord

На рисунке 2.2 представлена функция создания записи.

Поблочное описание функции:

Блок 1 – выделяет память под новую запись (структуру MaterialRecord).

Блок 2 – устанавливает поле next структуры record в значение NULL.

Это гарантирует, что поле next указывает на конец списка или связанной структуры данных.

Блок 3 – возвращает указатель record на созданную запись (структуру MaterialRecord).

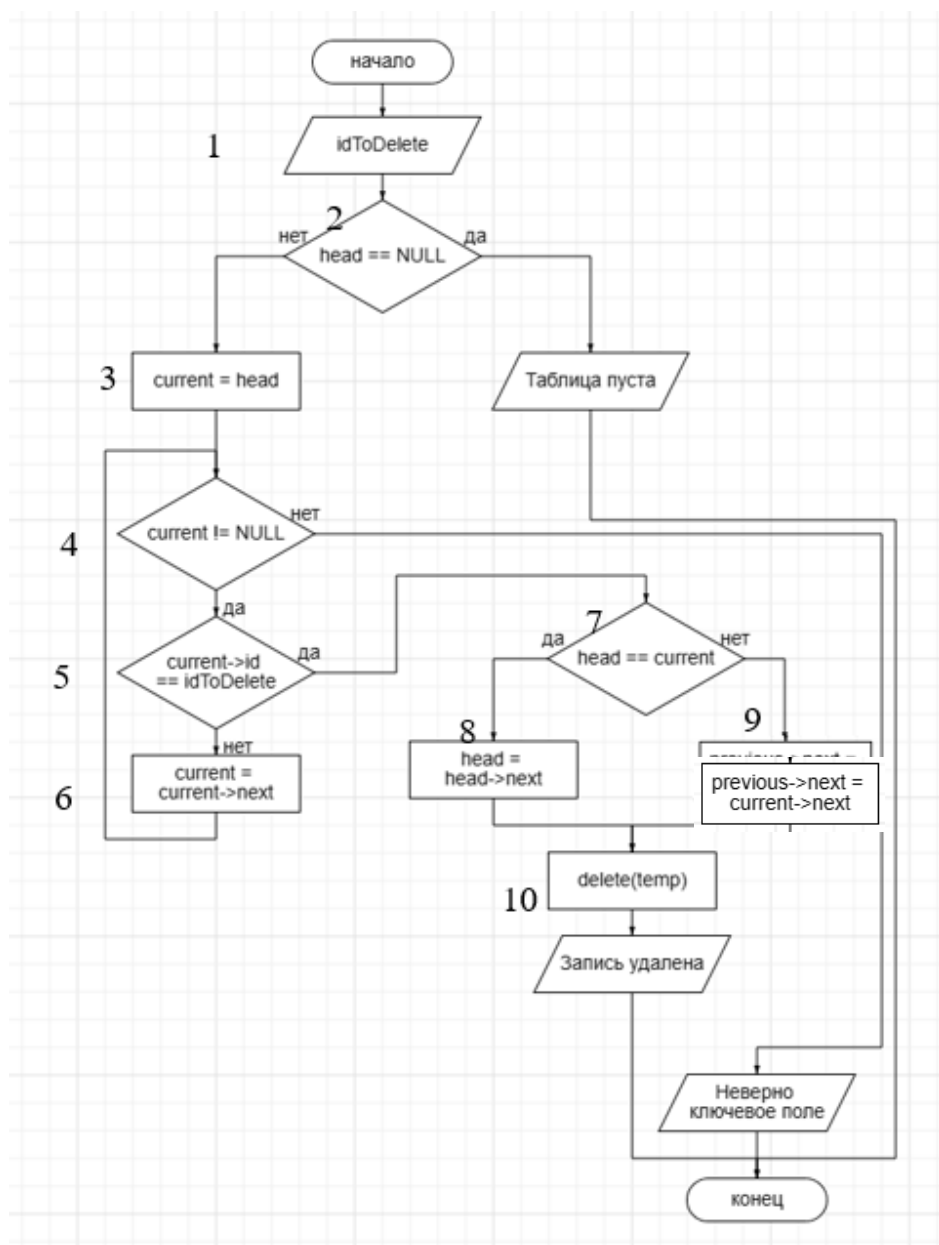


Рисунок 2.3 – Алгоритм функции deleteRecord

На рисунке 2.3 представлена функция удаления записи из таблицы.

Поблочное описание функции:

Блок 1 – ввод ключевого поля (порядкового номера), по которому будет удаляться поле

Блок 2 – проверяет, если переменная head равна NULL. Если head равен NULL, это означает, что таблица пуста, и выводится сообщение "Таблица пуста."

Блок 3 – создает указатель current на структуру Record и присваивает ему значение head. Current используется для прохода по связному списку и поиска записи по ключевому полю.

Блок 4 – запускает цикл while, который выполняется, пока current не равно NULL. Цикл используется для поиска записи с указанным ключевым полем в связном списке.

Блок 5 – проверяет, если значение ключевого поля текущей записи current равно введенному пользователем значению idToDelete.

Блок 6 – обновляет переменную current, присваивая ей значение указателя next текущей записи current. Таким образом, текущая запись переходит к следующей записи в связном списке.

Блок 7 – Если current равен head, это означает, что текущая запись current является первой записью в списке.

Блок 8 – обновляет переменную head, присваивая ей значение указателя next текущей записи current. Таким образом, первая запись в списке изменяется на следующую запись.

Блок 9 – обновляет указатель next предыдущей записи current, присваивая ему значение указателя next текущей записи current. Таким образом, связь между предыдущей и следующей записями обновляется, пропуская текущую запись.

Блок 10 – освобождает память, занятую текущей записью current, с помощью функции delete().

На рисунке 2.4 представлена функция корректировки записи в таблице.

Поблочное описание функции:

Блок 1 – ввод ключевого поля (порядковый номер) для корректировки записи

Блок 2 – создает указатель current на структуру MaterialRecord и присваивает ему значение head. current используется для прохода по связному списку и поиска записи по ключевому полю.

Блок 3 – запускает цикл while, который выполняется, пока current не равно NULL. Цикл используется для поиска записи с указанным ключевым полем в связном списке.

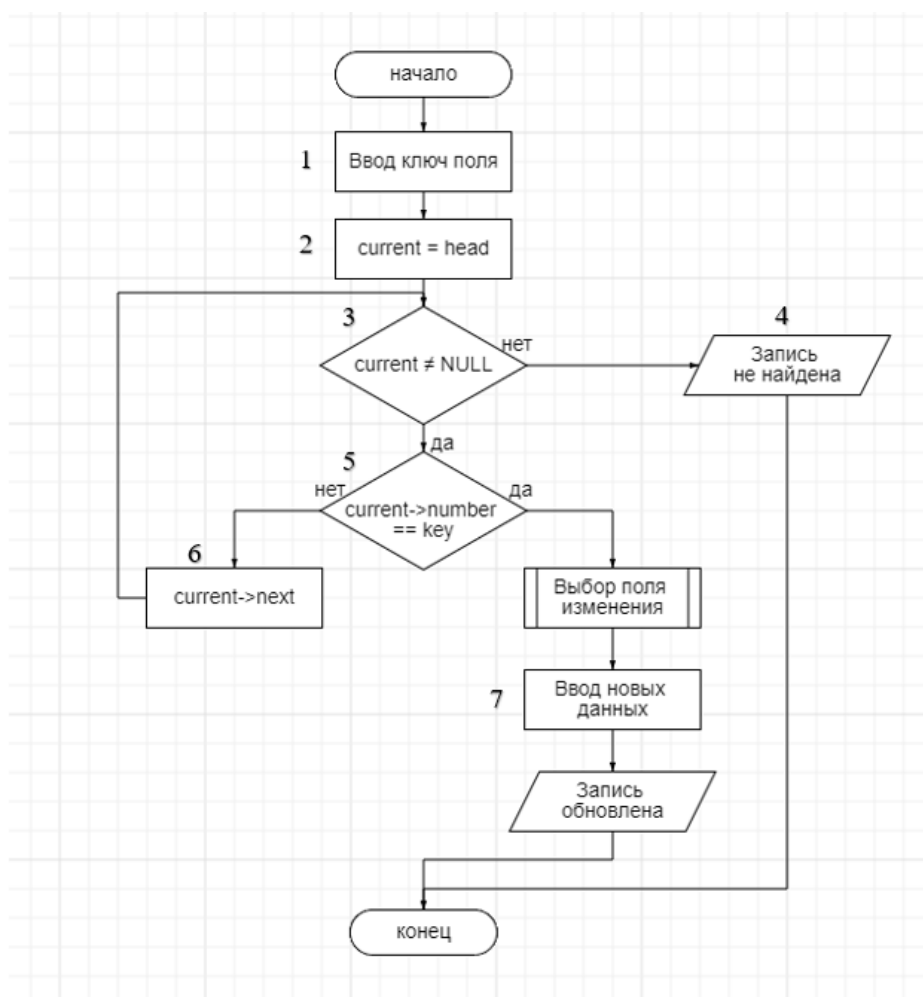


Рисунок 2.4 – Алгоритм функции editRecord

Блок 4 – Запись с указанным ключевым полем не найдена. После чего окончания выполнения операции.

Блок 5 – проверяет, если значение ключевого поля текущей записи current равно введенному пользователем значению key(idToEdit).

Блок 6 – перемещает указатель current на следующую запись в связном списке.

Блок 7 – ввод новых данных записи: номер завода, номер филиала, фамилия ответственного за материальные ценности, наличие материальных ценностей на начало периода, получено материальных ценностей на сумму, выбыло на сумму.



Рисунок 2.5 – Функция searchRecord

На рисунке 2.5 представлена функция поиска записи в таблице.

Поблочное описание функции:

Блок 1 – ввод ключевого поля (Фамилия) для поиска записи.

Блок 2 – создает указатель `current` на структуру `Record` и присваивает ему значение `head`. `current` используется для прохода по связному списку и поиска записи по указанной фамилии.

Блок 3 – объявляет переменную `found` и инициализирует ее значением 0. Эта переменная будет использоваться для отслеживания того, была ли найдена запись с указанной фамилией.

Блок 4 – запускает цикл `while`, который выполняется, пока `current` не равно `NULL`. Цикл используется для поиска записи с указанной фамилией в связном списке.

Блок 5 – сравнивает фамилию текущей записи `current->LastName` с введенной пользователем фамилией `searchName` с помощью функции `strcmp()`. Если результат сравнения равен 0, это означает, что фамилии совпадают и запись найдена.

Блок 6 – вывод найденной записи о материальных ценностях.

Блок 7 – устанавливает переменную `found` в значение 1, чтобы указать, что запись была найдена.

Блок 8 – перемещает указатель `current` на следующую запись в связном списке.

Блок 9 – проверяет значение переменной `found`. Если `found` равно 0, то выводится сообщение на экран, указывающее, что запись с указанной фамилией не была найдена.

На рисунке 2.6 представлена функция сохранения таблицы в типизированный файл.

Поблочное описание функции:

Блок 1 – считывает имя файла с клавиатуры и сохраняет ее в переменную filename.

Блок 2 – объявляет указатель на тип FILE с именем file и открывает файл с именем, указанным в filename, в режиме записи бинарных данных ("wb"). Если открытие файла не удалось, указатель file будет равен NULL.

Блок 3 – проверяет, удалось ли открыть файл. Если указатель file равен NULL, это означает, что открытие файла не удалось.

Блок 4 – объявляет указатель current на структуру MaterialRecord и устанавливает его в начало связного списка записей (head).

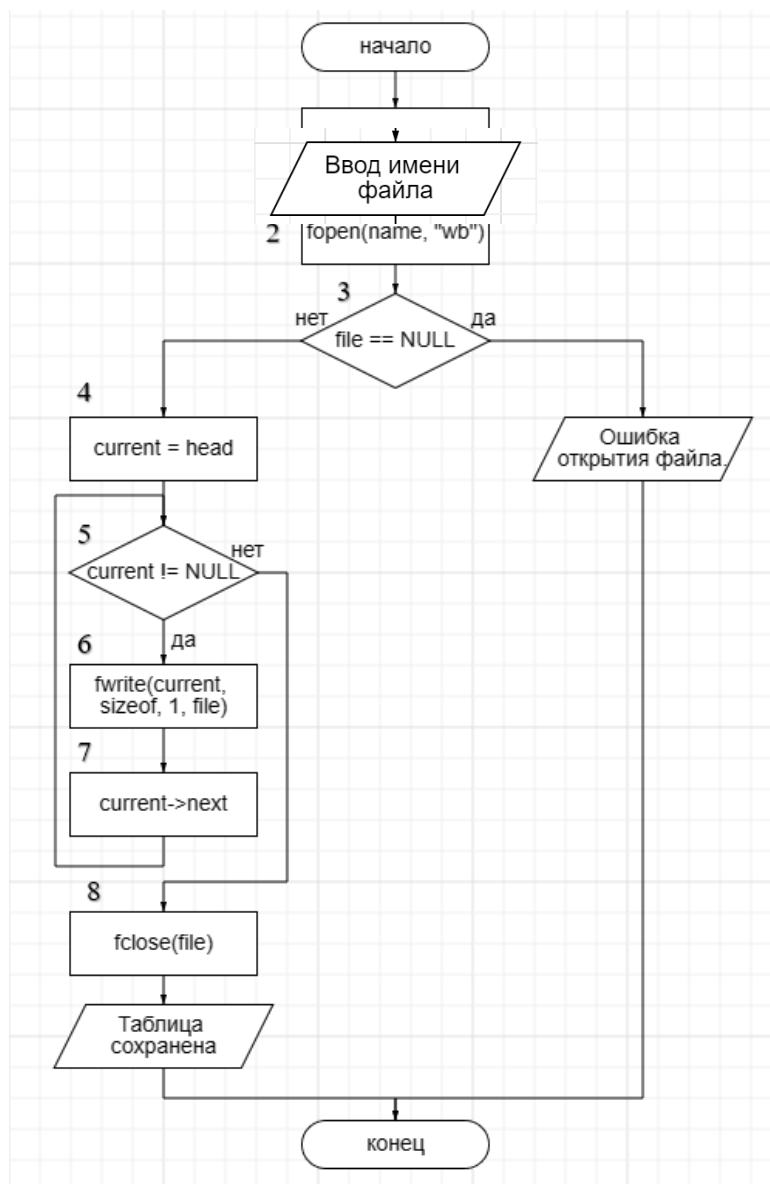


Рисунок 2.6 – Функция saveToBinaryFile

Блок 4 – объявляет указатель `current` на структуру `MaterialRecord` и устанавливает его в начало связанного списка записей (`head`).

Блок 5 – начинает цикл `while`, который будет выполняться, пока `current` указывает на действительную запись (не равную `NULL`).

Блок 6 – использует функцию `fwrite()` для записи содержимого структуры `current` в файл `file`. `sizeof(struct MaterialRecord)` указывает размер структуры `MaterialRecord`, а `1` указывает, что нужно записать одну структуру.

Блок 7 – переходит к следующей записи в связанном списке, обновляя указатель `current`.

Блок 8 – закрывает файл после сохранения таблицы.

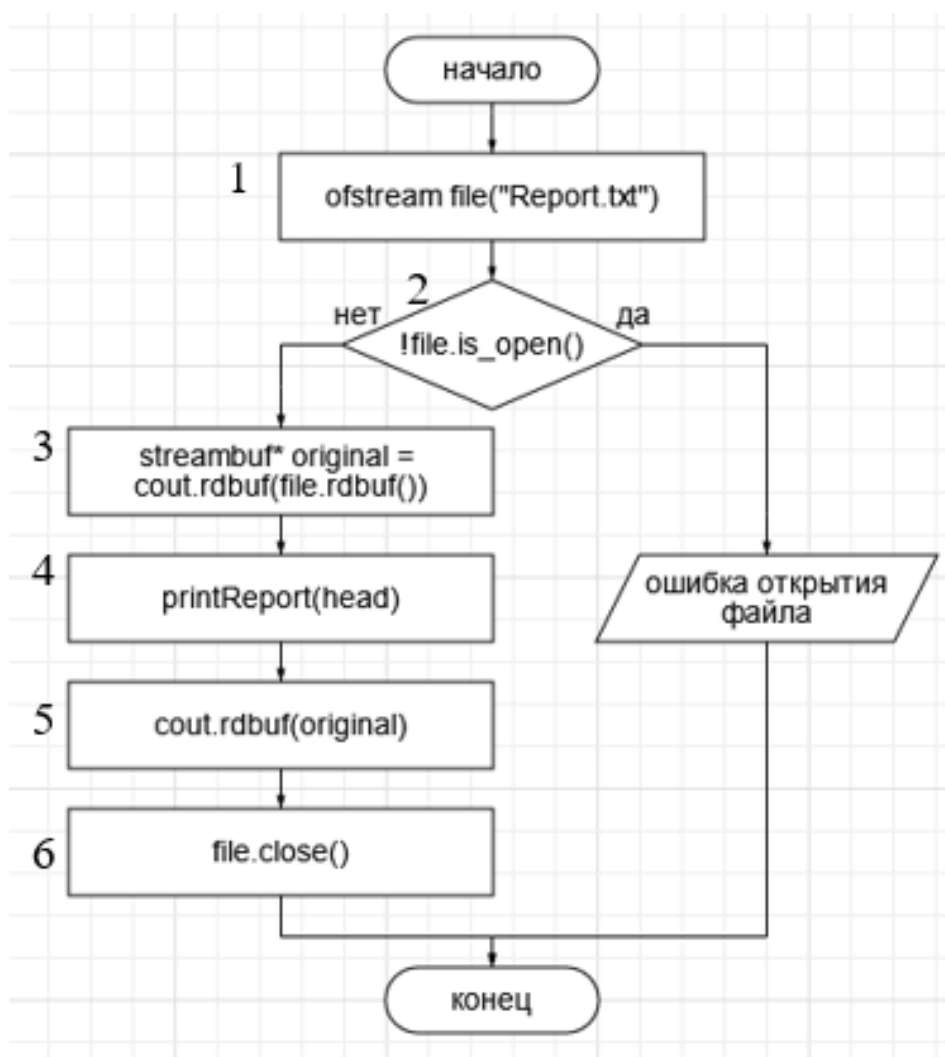


Рисунок 2.7 – Функция `printReportToFile`

На рисунке 2.7 представлена функция записи отчёта в файл.

Поблочное описание функции:

Блок 1 – открывает файл для записи отчёта.

Блок 2 – проверяет, удалось ли открыть файл для записи.

Блок 3 – перенаправляет поток вывода cout в файловый поток.

Блок 4 – вызывает функцию печати отчёта.

Блок 5 – восстанавливает исходный буфер для стандартного вывода cout.

Блок 6 – закрывает файл.

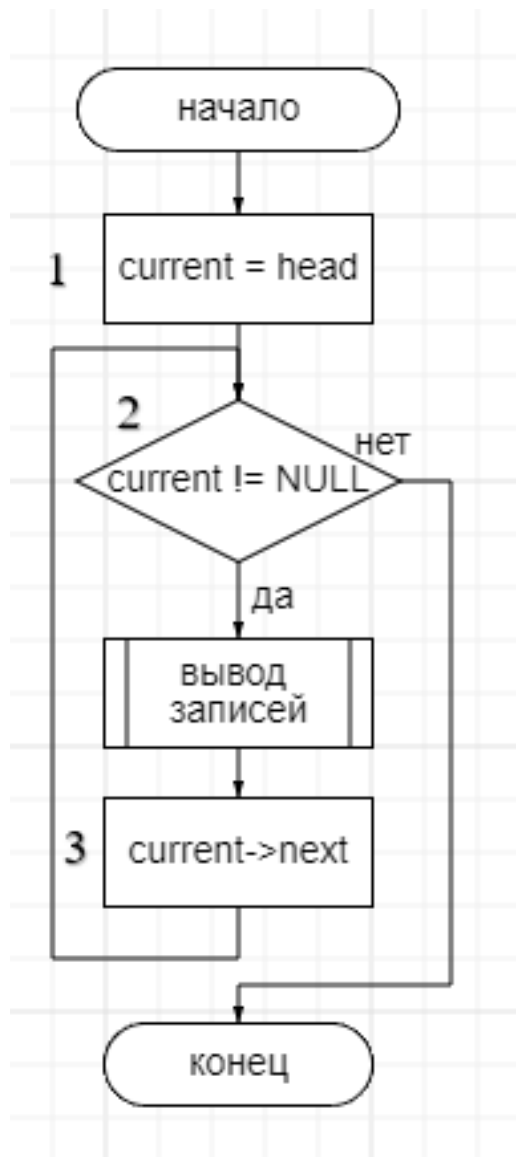


Рисунок 2.8 – Функция displayTable

На рисунке 2.8 представлена функция отображения данных в таблице.

Поблочное описание функции:

Блок 1 – объявляет указатель `current` и устанавливает его в начало связанного списка записей.

Блок 2 – начинает цикл `while`, который будет выполняться, пока `current` указывает на действительную запись (не равную `NULL`).

Блок 3 – переходит к следующей записи в связанном списке, обновляя указатель `current`.

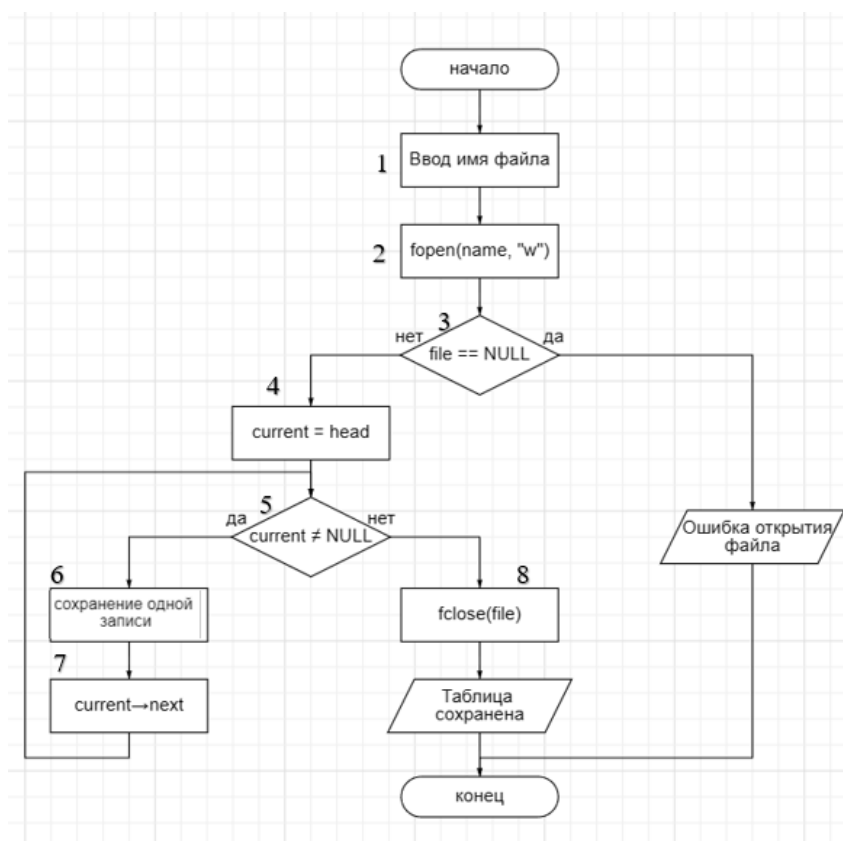


Рисунок 2.9 – Функция `saveToTextFile`

На рисунке 2.9 представлена функция сохранения таблицы в текстовый файл.

Поблочное описание функции:

Блок 1 – считывает введенное пользователем имя файла и сохраняет его в массив символов `filename`.

Блок 2 – открывает файл с именем, указанным в `filename`, в режиме записи ("w"). Если файл не существует, он будет создан.

Блок 3 – проверяет, удалось ли открыть файл для записи. Если file равен NULL, это означает, что открытие файла не удалось.

Блок 4 – создает указатель current на структуру Record и присваивает ему значение head. current используется для прохода по связному списку и сохранения каждой записи в файле.

Блок 5 – запускает цикл while, который выполняется, пока current не равно NULL. Цикл используется для сохранения каждой записи в файле.

Блок 6 – записывает значения полей каждой записи в файл с помощью функции fprintf(). Форматированная строка определяет порядок и типы данных, которые будут записаны в файл.

Блок 7 – перемещает указатель current на следующую запись в связном списке.

Блок 8 – закрывает файл после сохранения таблицы.

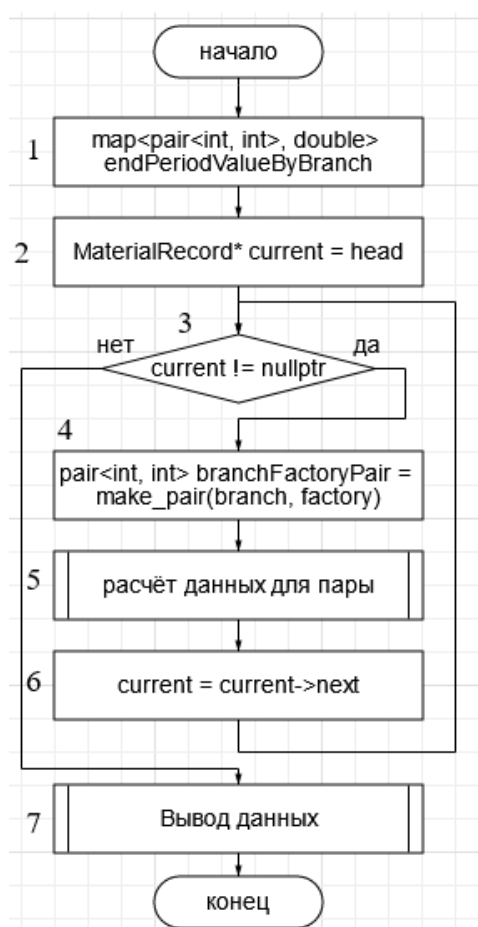


Рисунок 2.10 – Функция calculateEndPeriodValueByBranch

На рисунке 2.10 представлена функция сохранения таблицы в текстовый файл.

Поблочное описание функции:

Блок 1 – Инициализирует пустую карту endPeriodValueByBranch.

Блок 2 – создает указатель current на структуру Record и присваивает ему значение head. current используется для прохода по связному списку и сохранения каждой записи в файле.

Блок 3 – запускает цикл while, который выполняется, пока current не равно NULL.

Блок 4 – создает пару, содержащую номер фабрики и филиала.

Блок 5 – считает итоговое значение материальных ценностей по каждому филиалу.

Блок 6 – перемещает указатель current на следующую запись в связном списке.

Блок 7 – выводит обработанные данные пользователю.

2.5 Обоснование состава технических и программных средств

Технические средства:

Для запуска программы необходим персональный компьютер с операционной системой Windows.

Минимальные требования к аппаратному обеспечению включают процессор с тактовой частотой не менее 1 ГГц, 1 ГБ оперативной памяти, и несколько мегабайт свободного места на жестком диске.

Программные средства:

Операционная система Windows.

Исходный код программы: Весь исходный код программы, включая заголовочные файлы и текст программы на языке C, предоставляется в виде исходных файлов.

Библиотеки: для работы с файлами и вводом-выводом используются стандартные библиотеки языка C, такие как `stdio.h`, `stdlib.h`, и другие.

Инструменты разработки: для разработки программы использовалась среда Embarcadero Dev-C++. Она предоставляет удобную платформу для разработки, отладки и сборки программы.

Тестовые данные: для тестирования программы могут использоваться тестовые данные, созданные пользователем.

Все указанные программные и технические средства обоснованы на основе анализа требований программы, а также удовлетворяют минимальным системным требованиям, необходимым для нормального функционирования программы "Учет и управление данными о материальных ценностях" на платформе Windows.

Размер exe-файла программы – 2,4 МБ

Размер src-файла программы – 40 КБ

Размер одной сохраненной записи в текстовом файле – 32 Б

Размер одной сохраненной записи в типизированном файле – 72 Б

3. ВЫПОЛНЕНИЕ ПРОГРАММЫ

3.1 Условия выполнения программы

Минимальный состав аппаратных средств:

- Персональный компьютер (ПК) под управлением операционной системы Windows.
- Процессор с тактовой частотой не менее 1 ГГц.
- Минимум 1 ГБ оперативной памяти (RAM).
- Свободное место на жестком диске для хранения программы и данных.
- Клавиатура и мышь (или другие устройства ввода).

Минимальное программное обеспечение:

- Операционная система Windows (XP, 7, 8, 10 и др.).
- Компилятор языка C, такой как GCC, для сборки программы.

3.2 Загрузка и запуск программы

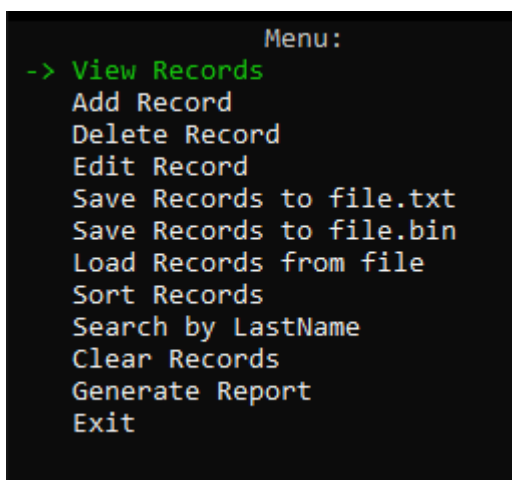
Инсталляция программы: Установка программы не требуется, поскольку она поставляется в виде исходного кода и может быть скомпилирована с использованием компилятора языка C. Пользователь должен убедиться в наличии необходимого компилятора и инструментов разработки, таких как Microsoft Visual Studio или Embarcadero Dev-C++, для компиляции программы из исходного кода.

Загрузка и компиляция:

- Скачайте исходный код программы из репозитория или хранилища.
- Откройте исходный код программы в выбранной среде разработки (например, Microsoft Visual Studio).
- Скомпилируйте программу, следуя инструкциям вашей среды разработки.

Запуск программы:

- После успешной компиляции запустите программу, следуя инструкциям вашей среды разработки.
- Программа будет выполняться в командной строке, предоставляя пользователю меню с доступными функциями.



```
Menu:
-> View Records
Add Record
Delete Record
Edit Record
Save Records to file.txt
Save Records to file.bin
Load Records from file
Sort Records
Search by LastName
Clear Records
Generate Report
Exit
```

Рисунок 3.1 – Меню программы с реализованным переключением функций и их выбором

Завершение программы: Программа может быть завершена пользователем, выбрав соответствующую опцию в меню программы.

Экранные формы и сообщения: Программа взаимодействует с пользователем через текстовую консоль, предоставляя информацию и инструкции на экране. Пользователю будут предоставлены соответствующие

сообщения и запросы для выполнения различных операций, таких как добавление, удаление, корректировка и просмотр записей.

На рисунках ниже показан процесс тестирования программы и работоспособность каждого из пунктов меню.

“Add Record”: при выборе данного пункта начнётся создание записи о клерке – пользователь осуществляет ввод данных.

```
Enter Factory Number: 3
Enter Branch Number: 234
Enter Last Name: Smith
Enter Start Value: 234
Enter Received Value: 43
Enter Disposed Value: 240
```

Рисунок 3.2 – Пример создания записи о клерке

“View Records”: при выборе данного пункта пользователь может ознакомиться со всеми записями в файле.

Current Page: 1

#	ID	Factory	Branch	Last Name	Start Value	Received Value	Disposed Value
1	1	1	101	Smith	5000.00	1000.00	500.00
2	2	2	101	Smith	7000.00	1500.00	800.00
3	3	2	201	Williams	6000.00	1200.00	700.00
4	4	2	101	Smith	400.00	1500.00	100.00
5	5	1	102	Besson	70000.00	1100.00	0.00
6	6	1	101	Smith	5000.00	1000.00	500.00
7	7	2	101	Smith	7000.00	1500.00	800.00
8	8	2	201	Williams	6000.00	1200.00	700.00
9	9	2	101	Smith	400.00	1500.00	100.00
10	10	1	102	Besson	70000.00	1100.00	0.00
11	11	1	101	Smith	5000.00	1000.00	500.00
12	12	2	101	Smith	7000.00	1500.00	800.00
13	13	2	201	Williams	6000.00	1200.00	700.00
14	14	2	101	Smith	400.00	1500.00	100.00
15	15	1	102	Besson	70000.00	1100.00	0.00
16	16	1	101	Smith	5000.00	1000.00	500.00
17	17	2	101	Smith	7000.00	1500.00	800.00
18	18	2	201	Williams	6000.00	1200.00	700.00
19	19	2	101	Smith	400.00	1500.00	100.00
20	20	1	102	Besson	70000.00	1100.00	0.00

Press 'w' for previous page, 's' for next page, 'Enter' to exit:

Рисунок 3.3 – Просмотр всех записей в порядке их записи

“Sort Records”: при выборе данного пункта программа предоставит выбор пользователю касаясь ключевого поля и направлению сортировки, по которому в дальнейшем будут отсортированы данные в таблице.

Current Page: 1

#	ID	Factory	Branch	Last Name	Start Value	Received Value	Disposed Value
1	50	1	102	Besson	7000.00	1100.00	0.00
2	49	2	101	Smith	400.00	1500.00	100.00
3	48	2	201	Williams	6000.00	1200.00	700.00
4	47	2	101	Smith	7000.00	1500.00	800.00
5	46	1	101	Smith	5000.00	1000.00	500.00
6	45	1	102	Besson	7000.00	1100.00	0.00
7	44	2	101	Smith	400.00	1500.00	100.00
8	43	2	201	Williams	6000.00	1200.00	700.00
9	42	2	101	Smith	7000.00	1500.00	800.00
10	41	1	101	Smith	5000.00	1000.00	500.00
11	40	1	102	Besson	7000.00	1100.00	0.00
12	39	2	101	Smith	400.00	1500.00	100.00
13	38	2	201	Williams	6000.00	1200.00	700.00
14	37	2	101	Smith	7000.00	1500.00	800.00
15	36	1	101	Smith	5000.00	1000.00	500.00
16	35	1	102	Besson	7000.00	1100.00	0.00
17	34	2	101	Smith	400.00	1500.00	100.00
18	33	2	201	Williams	6000.00	1200.00	700.00
19	32	2	101	Smith	7000.00	1500.00	800.00
20	31	1	101	Smith	5000.00	1000.00	500.00

Press 'w' for previous page, 's' for next page, 'Enter' to exit:

Рисунок 3.4 – Результат сортировки записей по id по убыванию

“Delete Record”: при выборе данного пункта пользователь может удалить запись по введённому порядковому номеру.

```
Record with ID 4 has been deleted.
-->Press any key to go back<--
```

Рисунок 3.5 – Пример удаления записи о клерке

“Edit Record”: при выборе данного пункта пользователь может скорректировать данные в записи по введённому порядковому номеру.

Edit Record 1

ID	Factory	Branch	Last Name	Start Value	Received Value	Disposed Value
1	1	101	Smith	5000.00	1000.00	500.00

-> Edit Factory Number
 Edit Branch Number
 Edit Last Name
 Edit Start Value
 Edit Received Value
 Edit Disposed Value
 Edit All Attributes
 Back To Main Menu

Рисунок 3.6 – Пример корректировки записи о клерке по ключевому полю (порядковый номер)

“Search by LastName”: при выборе данного пункта пользователь может узнать всю информацию по записям данного человека.

Enter '0' to go to the main menu
Enter Last Name to search: Smith
Search results for 'Smith':

ID	Factory	Branch	Last Name	Start Value	Received Value	Disposed Value
1	1	101	Smith	5000.00	1000.00	500.00
2	2	101	Smith	7000.00	1500.00	800.00
4	2	101	Smith	400.00	1500.00	100.00
6	1	101	Smith	5000.00	1000.00	500.00
7	2	101	Smith	7000.00	1500.00	800.00
9	2	101	Smith	400.00	1500.00	100.00
11	1	101	Smith	5000.00	1000.00	500.00
12	2	101	Smith	7000.00	1500.00	800.00
14	2	101	Smith	400.00	1500.00	100.00
16	1	101	Smith	5000.00	1000.00	500.00
17	2	101	Smith	7000.00	1500.00	800.00
19	2	101	Smith	400.00	1500.00	100.00
21	1	101	Smith	5000.00	1000.00	500.00
22	2	101	Smith	7000.00	1500.00	800.00
24	2	101	Smith	400.00	1500.00	100.00
26	1	101	Smith	5000.00	1000.00	500.00
27	2	101	Smith	7000.00	1500.00	800.00
29	2	101	Smith	400.00	1500.00	100.00
31	1	101	Smith	5000.00	1000.00	500.00
32	2	101	Smith	7000.00	1500.00	800.00
34	2	101	Smith	400.00	1500.00	100.00
36	1	101	Smith	5000.00	1000.00	500.00
37	2	101	Smith	7000.00	1500.00	800.00
39	2	101	Smith	400.00	1500.00	100.00
41	1	101	Smith	5000.00	1000.00	500.00
42	2	101	Smith	7000.00	1500.00	800.00
44	2	101	Smith	400.00	1500.00	100.00
46	1	101	Smith	5000.00	1000.00	500.00
47	2	101	Smith	7000.00	1500.00	800.00
49	2	101	Smith	400.00	1500.00	100.00

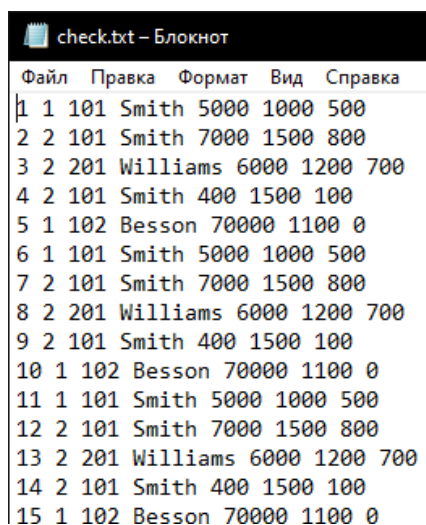
-->Press any key to go back<--

Рисунок 3.7 – Пример поиска записи по фамилии ответственного
“Generate Report”: при выборе данного пункта программа предлагает
выбрать какие именно данные надо обработать.

REPORT						
Factory	Branch	Start Value	Received Value	Disposed Value	End Value	
1	101	50000.00	10000.00	5000.00	55000.00	
2	101	74000.00	30000.00	9000.00	95000.00	
1	102	700000.00	11000.00	0.00	711000.00	
2	201	60000.00	12000.00	7000.00	65000.00	
Factory		Start Value	Received Value	Disposed Value	End Value	
1		750000.00	21000.00	5000.00	766000.00	
2		134000.00	42000.00	16000.00	160000.00	
Total:		884000.00	63000.00	21000.00	926000.00	
-->Press any key to go back<--						

Рисунок 3.8 – Вывод отчёта

“Save Record to *.txt”: при выборе данного пункта программа создаст
пользователю текстовый файл, содержащий все данные из таблицы.



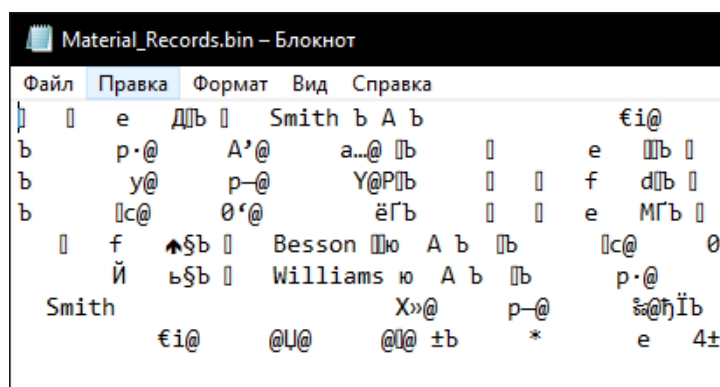
```

1 1 101 Smith 5000 1000 500
2 2 101 Smith 7000 1500 800
3 2 201 Williams 6000 1200 700
4 2 101 Smith 400 1500 100
5 1 102 Besson 70000 1100 0
6 1 101 Smith 5000 1000 500
7 2 101 Smith 7000 1500 800
8 2 201 Williams 6000 1200 700
9 2 101 Smith 400 1500 100
10 1 102 Besson 70000 1100 0
11 1 101 Smith 5000 1000 500
12 2 101 Smith 7000 1500 800
13 2 201 Williams 6000 1200 700
14 2 101 Smith 400 1500 100
15 1 102 Besson 70000 1100 0

```

Рисунок 3.9 – Сохранение в текстовый файл всех данных

“Save Record to *.bin”: при выборе данного пункта программа создаст пользователю бинарный файл, содержащий все данные из таблицы.



```

e ДПб  Smith б А б €i@
б р·@ А'@ а...@ Пб  e Пб  |
б у@ р-@ Y@PПб  |  | f dПб  |
б |с@ 0'@ ёГб  |  | e МГб  |
| f ▲$б  Besson Пю А б Пб  |с@ 0
Й ь$б  Williams ю А б Пб  р·@
Smith X»@ р-@ ь@hПб
€i@ @П@ @П@ ±б * e 4±

```

Рисунок 3.10 – Сохранение в типизированный файл всех данных

“Load Records from file”: при выборе данного пункта пользователю нужно будет ввести имя файла (без формата), который впоследствии программа скопирует в таблицу данных.

“Clear Records”: при выборе данного пункта программа очищает все существующие записи.

“Exit”: при выборе данного пункта пользователем программа закрывается, очищая полностью весь буфер данных.

Пользователь может также вводить тестовые данные и проверять работу программы на реальных или смоделированных данных.

ЗАКЛЮЧЕНИЕ

В ходе разработки программы для управления записями о материальных ценностях были выполнены задачи, определенные в техническом задании. Получены следующие результаты:

- Разработана программа, позволяющая пользователю создавать, удалять, корректировать, сортировать и искать записи о сотрудниках.
- Программа предоставляет удобный интерфейс в виде текстовой консоли и позволяет выполнять все перечисленные операции взаимодействия с данными.
- Процедура сохранения и загрузки таблицы из файла позволяет пользователям сохранять свои данные и восстанавливать их после перезапуска программы.
- После обработки данных программа выводит результаты на экран и сохраняет их в текстовом файле для дальнейшего анализа.

Данная программа имеет потенциал использования в различных областях, связанных с управлением данными о материальных ценностях. Она может быть полезна в управлении фабриками, заводами, филиалами

Цель проекта была успешно достигнута. Полученный результат позволяет пользователям удобно и эффективно взаимодействовать с данными и обеспечивает сохранность информации.

Рекомендации по дальнейшему использованию программы включают в себя поддержку и обновление программы, а также возможное расширение ее функциональности в соответствии с потребностями конкретной области применения.

Разработка данной программы оказалась успешной и соответствует поставленным задачам и целям проекта.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Конова, Е. А. Алгоритмы и программы. Язык С++ / Е. А. Конова, Г. А. Поллак. — 7-е изд., стер. — Санкт-Петербург : Лань, 2023. — 384 с. — ISBN 978-5-507-46070-0. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/297002> (дата обращения: 23.11.2023). — Режим доступа: для авториз. пользователей.
2. Мурлин, А. Г. Объектно-ориентированное программирование : учебное пособие / А. Г. Мурлин, В. А. Мурлина, М. В. Янаева. — Краснодар : КубГТУ, 2021. — 151 с. — ISBN 978-5-8333-1059-5. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/231569> (дата обращения: 23.11.2023). — Режим доступа: для авториз. пользователей.
3. Скворцова, Л. А. Объектно-ориентированное программирование на языке С++: Практикум : учебное пособие / Л. А. Скворцова, А. А. Бирюкова, К. В. Гусев. — Москва : РТУ МИРЭА, 2021. — 146 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/176540> (дата обращения: 23.11.2023). — Режим доступа: для авториз. пользователей.
4. Теплоухов, С. В. Основы объектно-ориентированного программирования на языке С++ : учебное пособие / С. В. Теплоухов. — Майкоп : АГУ, 2021. — 92 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/231416> (дата обращения: 23.11.2023). — Режим доступа: для авториз. пользователей.
5. Конова, Е. А. Алгоритмы и программы. Язык С++ : учебное пособие для вузов / Е. А. Конова, Г. А. Поллак. — 6-е изд., стер. — Санкт-Петербург : Лань, 2021. — 384 с. — ISBN 978-5-8114-8487-4. — Текст :

электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/176900> (дата обращения: 23.11.2023). — Режим доступа: для авториз. пользователей.

6. Скворцова, Л. А. Объектно-ориентированное программирование на языке C++ : учебное пособие / Л. А. Скворцова. — Москва : РТУ МИРЭА, 2020. — 246 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/163862> (дата обращения: 23.11.2023). — Режим доступа: для авториз. пользователей.

7. Зайцев, М. Г. Объектно-ориентированный анализ и программирование : учебное пособие / М. Г. Зайцев. — Новосибирск : НГТУ, 2017. — 84 с. — ISBN 978-5-7782-3308-9. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/118271> (дата обращения: 23.11.2023). — Режим доступа: для авториз. пользователей.

8. Скворцова, Л. А. Объектно-ориентированное программирование на языке C++: Практикум : учебное пособие / Л. А. Скворцова, А. А. Бирюкова, К. В. Гусев. — Москва : РТУ МИРЭА, 2021. — 146 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/176540> (дата обращения: 23.11.2023). — Режим доступа: для авториз. пользователей.

9. Теплоухов, С. В. Основы объектно-ориентированного программирования на языке C++ : учебное пособие / С. В. Теплоухов. — Майкоп : АГУ, 2021. — 92 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/231416> (дата обращения: 23.11.2023). — Режим доступа: для авториз. пользователей.

10. Карпенко, С. Н. Основы объектно-ориентированного программирования на языке C++ : учебно-методическое пособие / С. Н. Карпенко. — Нижний Новгород : ННГУ им. Н. И. Лобачевского, 2018. — 104

с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/book/144808> (дата обращения: 23.11.2023). — Режим доступа: для авториз. пользователей.

Эти источники были использованы для получения информации о функциях и библиотеках, используемых в программе, а также для обеспечения правильной реализации стандартных операций в С.

ПРИЛОЖЕНИЕ А

ОФОРМЛЕНИЕ ТЕКСТА ПРОГРАММЫ

```

#include <iostream>
#include <fstream>
#include <string>
#include <conio.h>
#include <windows.h>
#include <iomanip>
#include <map>

#define KEY_W 119
#define KEY_S 115
#define KEY_UP 72
#define KEY_DOWN 80
#define KEY_ENTER 13
#define KEY_ESC 27

using namespace std;

//#####BASEme
nt#####//

struct MaterialRecord {
    int id;
    int factoryNumber;
    int branchNumber;
    string LastName;

```

```

    double startValue;
    double receivedValue;
    double disposedValue;
    MaterialRecord* next;
};

MaterialRecord* createRecord(int factoryNumber, int
branchNumber, const string& lastName, double startValue,
double receivedValue, double disposedValue) {
    static int currentId = 1; // Статическая переменная
для отслеживания текущего значения id
    MaterialRecord* newRecord = new MaterialRecord;

    newRecord->id = currentId++;
    newRecord->factoryNumber = factoryNumber;
    newRecord->branchNumber = branchNumber;
    newRecord->LastName = lastName;
    newRecord->startValue = startValue;
    newRecord->receivedValue = receivedValue;
    newRecord->disposedValue = disposedValue;
    newRecord->next = nullptr;

    return newRecord;
}

//#####Prototy
pe functions#####//

```

```

void saveToTextFile(MaterialRecord* head, const string&
filename);
void saveToBinaryFile(MaterialRecord* head, const
string& filename);
void loadFromTextFile(MaterialRecord*& head, const
string& filename);

void calculateEndPeriodValueByBranch(MaterialRecord*
head);
void calculateEndPeriodValueByFactory(MaterialRecord*
head);
void calculateTotals(MaterialRecord* head);

void displayRecords(MaterialRecord* head);

void sortRecords(MaterialRecord*& head);

void releaseMemory(MaterialRecord*& head);

void printReport(MaterialRecord* head);
void printReportToFile(MaterialRecord* head);

//#####Record
handing#####//

void addRecord(MaterialRecord*& head, MaterialRecord*
newRecord) {
    if (head == nullptr) {
        head = newRecord;
    }
}

```

```

    } else {
        MaterialRecord* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newRecord;
    }
    system("cls");
    cout << "Record added successfully!" << endl;
}

void displayRecords(MaterialRecord* head) {
    int currentPage = 1;
    int itemsPerPage = 20;
    char key = 0;
    MaterialRecord* current = head;

    do {
        system("cls");
        printf("Current Page: %d\n", currentPage);
        cout << "-----
-----" << endl;

        cout << "|  #   |   ID   | Factory | Branch |
Last Name   | Start Value | Received Value | Disposed
Value |" << endl;
        cout << "-----
-----" << endl;
    }

```

```

        int count = 0;
        int lineNumber = 1 ; // Начальный номер строки
        while (current != nullptr && count <
itemsPerPage) {
            printf("| %3d | %6d | %7d | %6d | %13s |
%12.2f | %14.2f | %14.2f |\n", lineNumber++, current-
>id, current->factoryNumber,
                    current->branchNumber,          current-
>LastName.c_str(),      current->startValue,          current-
>receivedValue,
                    current->disposedValue);
            current = current->next;
            count++;
        }

        printf("\nPress 'w' for previous page, 's' for
next page, 'Enter' to exit: ");
        key = getch();

        if (key == KEY_S || key == KEY_DOWN) {
            currentPage++;
            if (current == nullptr) {
                currentPage--;
            }
            MaterialRecord* temp = head;
            for (int i = 1; i < (currentPage - 1) *
itemsPerPage; i++) {
                temp = temp->next;
            }
        }
    }
}

```

```

    }
    current = temp;
} else if (key == KEY_W || key == KEY_UP) {
    currentPage--;
    if (currentPage < 1) {
        currentPage = 1;
    }
    MaterialRecord* temp = head;
    for (int i = 1; i < (currentPage - 1) *
itemsPerPage; i++) {
        temp = temp->next;
    }
    current = temp;
} else {
    currentPage = currentPage;
    MaterialRecord* temp = head;
    for (int i = 1; i < (currentPage - 1) *
itemsPerPage; i++) {
        temp = temp->next;
    }
    current = temp;
}
} while (key != KEY_ENTER);
}

```

```

void deleteRecord(MaterialRecord*& head, int
idToDelete) {
    if (idToDelete == 0) {
        system("cls");
    }
}

```

```

        return;
    }

    if (head == nullptr) {
        cout << "List is empty!" << endl;
        cout << "-->Press any key to go back<--";
        getch();
        system("cls");
        return;
    }

    if (head->id == idToDelete) {
        MaterialRecord* temp = head;
        head = head->next;
        delete temp;
        cout << "Record with ID " << idToDelete << "
has been deleted." << endl;
        cout << "-->Press any key to go back<--";
        getch();
        system("cls");
        return;
    }

    MaterialRecord* current = head;
    while (current->next != nullptr) {
        if (current->next->id == idToDelete) {
            MaterialRecord* temp = current->next;
            current->next = current->next->next;
            delete temp;

```



```

        cout << "Record with ID " << idToDelete <<
" has been deleted." << endl;
        cout << "-->Press any key to go back<--";
        getch();
        system("cls");
        return;
    }
    current = current->next;
}

    cout << "Record with ID " << idToDelete << " not
found!" << endl;
    cout << "-->Press any key to go back<--";
    getch();
    system("cls");
}

void editRecord(MaterialRecord*& head, int idToEdit) {
    MaterialRecord* current = head;
    bool found = false;
    if (idToEdit == 0) {
        system("cls");
        return;
    }
    while (current != nullptr) {
        if (current->id == idToEdit) {
            found = true;
            break;
        }
    }
}

```

```

        current = current->next;
    }

    if (!found) {
        cout << "Record with ID " << idToEdit << " not
found!" << endl;
        cout << "Enter '0' to go to the main
menu"<<endl<<"Please enter a new ID: ";
        cin >> idToEdit;
        system("cls");
        editRecord(head, idToEdit);
        return;
    }

    int key, choice_edit = 0;
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    string print_edit[8]= {"Edit Factory Number","Edit
Branch Number","Edit Last Name","Edit Start Value","Edit
Received Value",
                           "Edit Disposed Value","Edit
All Attributes", "Back To Main Menu"
                           };

    int int_print_edit = 8;
    while(true) {
        do {
            cout << "
Edit Record
" << idToEdit << endl;

```

```

        cout << "-----
-----" << endl;
        cout << "|      ID      | Factory | Branch |
Last Name  | Start Value | Received Value | Disposed
Value |" << endl;
        cout << "-----
-----" << endl;
        printf("| %6d | %7d | %6d | %13s | %12.2f
| %14.2f | %14.2f |\n", current->id, current-
>factoryNumber,
                current->branchNumber, current-
>LastName.c_str(), current->startValue, current-
>receivedValue, current->disposedValue);
        cout << "-----
-----" << endl;
        for(int j=0; j<int_print_edit; j++) {
            if(j==choice_edit) {
                SetConsoleTextAttribute(hConsole,
10);
                cout<<" -> "<<print_edit[j]<<endl;
            } else
                cout<<"      "<<print_edit[j]<<endl;
            SetConsoleTextAttribute(hConsole, 15);
        }
        key=getch();

```

```

        if(key==KEY_S          ||          key==KEY_DOWN)
choice_edit++;
        if(key==KEY_W || key==KEY_UP) choice_edit-
-;

        if(choice_edit<0) choice_edit=0;
        if(choice_edit>int_print_edit-1)
choice_edit=int_print_edit-1;
        system("cls");
    } while(key != KEY_ENTER);

    switch (choice_edit) {
        case 0: {
            cout << "Enter new Factory Number: ";
            cin >> current->factoryNumber;
            break;
        }
        case 1: {
            cout << "Enter new Branch Number: ";
            cin >> current->branchNumber;
            break;
        }
        case 2: {
            cout << "Enter new Last Name: ";
            cin.ignore();
            getline(cin, current->LastName);
            break;
        }
        case 3: {
            cout << "Enter new Start Value: ";

```

```
        cin >> current->startValue;
        break;
    }
    case 4: {
        cout << "Enter new Received Value: ";
        cin >> current->receivedValue;
        break;
    }
    case 5: {
        cout << "Enter new Disposed Value: ";
        cin >> current->disposedValue;
        break;
    }
    case 6: {
        cout << "Enter new Factory Number: ";
        cin >> current->factoryNumber;
        cout << "Enter new Branch Number: ";
        cin >> current->branchNumber;
        cout << "Enter new Last Name: ";
        cin.ignore();
        getline(cin, current->LastName);
        cout << "Enter new Start Value: ";
        cin >> current->startValue;
        cout << "Enter new Received Value: ";
        cin >> current->receivedValue;
        cout << "Enter new Disposed Value: ";
        cin >> current->disposedValue;
        break;
    }
```

```

        case 7: {
            return;
        }
    }
    system("cls");

    cout << "Record ID " << idToEdit << " updated
successfully." << endl;
    cout << "-->Press any key to go back<--";
    getch();
    system("cls");

}

}

//#####rabota
s failami#####//

void saveToTextFile(MaterialRecord* head, const string&
filename) {
    ofstream file(filename);
    if (!file.is_open()) {
        cout << "Unable to open file!" << endl;
        return;
    }

    MaterialRecord* current = head;
    while (current != nullptr) {

```

```

        file << current->id << ' ' << current-
>factoryNumber << ' ' << current->branchNumber << ' '
        << current->LastName << ' ' << current-
>startValue << ' ' << current->receivedValue
        << ' ' << current->disposedValue << endl;
        current = current->next;
    }

```

```

    file.close();
    cout << "Data saved to text file successfully!" <<
endl;
    cout << "-->Press any key to go back<--";
    getch();
    system("cls");
}

```

```

void saveToBinaryFile(MaterialRecord* head, const char*
filename) {
    FILE* file = fopen(filename, "wb"); // Открытие
файла для записи в бинарном режиме

```

```

    if (file == nullptr) {
        cerr << "Unable to open file for writing!" <<
endl;
        return;
    }

```

```

    MaterialRecord* current = head;
    while (current != nullptr) {

```

```

        fwrite(current,    sizeof(MaterialRecord),    1,
file); // Запись одной записи в файл
        current = current->next;
    }

    fclose(file); // Заккрытие файла
    cout << "Data saved to binary file successfully!"
<< endl;
    cout << "-->Press any key to go back<--";
    getch();
    system("cls");
}

void loadFromTextFile(MaterialRecord*& head, const
string& filename) {
    ifstream file(filename);
    if (!file.is_open()) {
        cout << "Unable to open file!" << endl;
        return;
    }
    if (head != nullptr) {
        int key,choice=1;
        HANDLE hConsole =
GetStdHandle(STD_OUTPUT_HANDLE);
        string print_download[4]= {"Do you want to
clear existing records?", "Yes", "No", "Back To Main
Menu"

};
        int int_print_download = 4;

```



```

do {
    for(int j=0; j<int_print_download; j++) {
        if(j==choice) {
            SetConsoleTextAttribute(hConsole,
10);

            cout<<"                                ->
"<<print_download[j]<<endl;
        } else
            cout<<"
"<<print_download[j]<<endl;
            SetConsoleTextAttribute(hConsole, 15);
        }
        key=getch();
        if(key==KEY_S || key==KEY_DOWN) choice++;
        if(key==KEY_W || key==KEY_UP) choice--;
        if(choice<1) choice=1;
        if(choice>int_print_download-1)
choice=int_print_download-1;
        system("cls");
    } while(key!= KEY_ENTER);

    switch (choice) {
        case 1: {
            releaseMemory(head);
            cout << "Existing records cleared." <<
endl;

            break;
        }
    }

```

```

        case 2: {
            cout << "Existing records kept." <<
endl;

            break;
        }
        case 3: {
            return;
        }
    }
    int id, factory, branch;
    string lastName;
    double start, received, disposed;

    while (file >> id >> factory >> branch >>
lastName >> start >> received >> disposed) {
        MaterialRecord*      newRecord      =      new
MaterialRecord;
        newRecord->id = id;
        newRecord->factoryNumber = factory;
        newRecord->branchNumber = branch;
        newRecord->LastName = lastName;
        newRecord->startValue = start;
        newRecord->receivedValue = received;
        newRecord->disposedValue = disposed;
        newRecord->next = nullptr;

        if (head == nullptr) {
            head = newRecord;
        } else {

```

```

        MaterialRecord* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newRecord;
    }
}

file.close();
cout << "Data loaded from text file
successfully!" << endl;
cout << "-->Press any key to go back<--";
getch();
system("cls");
}
}

//#####sortir
ovka#####//

bool compareByColumnUp(const MaterialRecord& a, const
MaterialRecord& b, int column) {
    switch (column) {
        case 1:
            return a.id > b.id;
        case 2:
            return a.factoryNumber > b.factoryNumber;
        case 3:
            return a.branchNumber > b.branchNumber;
    }
}

```

```

        case 4:
            return a.LastName > b.LastName;
        case 5:
            return a.startValue > b.startValue;
        case 6:
            return a.receivedValue > b.receivedValue;
        case 7:
            return a.disposedValue > b.disposedValue;
    }
}

bool compareByColumnDown(const MaterialRecord& a, const
MaterialRecord& b, int column) {
    switch (column) {
        case 1:
            return a.id < b.id;
        case 2:
            return a.factoryNumber < b.factoryNumber;
        case 3:
            return a.branchNumber < b.branchNumber;
        case 4:
            return a.LastName < b.LastName;
        case 5:
            return a.startValue < b.startValue;
        case 6:
            return a.receivedValue < b.receivedValue;
        case 7:
            return a.disposedValue < b.disposedValue;
    }
}

```

```
}
```

```
void swap(MaterialRecord* a, MaterialRecord* b) {
    int tempId = a->id;
    int tempFactory = a->factoryNumber;
    int tempBranch = a->branchNumber;
    string tempLastName = a->LastName;
    double tempStart = a->startValue;
    double tempReceived = a->receivedValue;
    double tempDisposed = a->disposedValue;

    a->id = b->id;
    a->factoryNumber = b->factoryNumber;
    a->branchNumber = b->branchNumber;
    a->LastName = b->LastName;
    a->startValue = b->startValue;
    a->receivedValue = b->receivedValue;
    a->disposedValue = b->disposedValue;

    b->id = tempId;
    b->factoryNumber = tempFactory;
    b->branchNumber = tempBranch;
    b->LastName = tempLastName;
    b->startValue = tempStart;
    b->receivedValue = tempReceived;
    b->disposedValue = tempDisposed;
}
```

```
void sortRecords(MaterialRecord*& head) {
```

```

HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);

if (head == nullptr || head->next == nullptr) {
    cout << "List has zero or one element. No need
for sorting." << endl;
    return;
}
string column_sort;
int key;
int column = 1;
string          print_sort[9]=          {"
Sort:", "ID", "Factory", "Branch", "Last      Name", "Start
Value", "Received Value", "Disposed Value", "Back to Main
Menu"};
int int_print_sort = 9;
do {
    for(int j=0; j<int_print_sort; j++) {
        if(j==column) {
            SetConsoleTextAttribute(hConsole, 10);
            cout<<" -> "<<print_sort[j]<<endl;
        } else
            cout<<"      "<<print_sort[j]<<endl;
        SetConsoleTextAttribute(hConsole, 15);
    }
    key=getch();
    if(key==KEY_S || key==KEY_DOWN) column++;
    if(key==KEY_W || key==KEY_UP) column--;
    if(column<1) column=1;
}

```

```
        if(column>int_print_sort-1)
column=int_print_sort-1;
        system("cls");
    } while(key!=KEY_ENTER);

switch (column) {
    case 1: {
        column_sort = "id";
        break;
    }
    case 2: {
        column_sort = "Factory Number";
        break;
    }
    case 3: {
        column_sort = "Brunch Number";
        break;
    }
    case 4: {
        column_sort = "LastName";
        break;
    }
    case 5: {
        column_sort = "Start Value";
        break;
    }
    case 6: {
        column_sort = "Received Value";
        break;
    }
}
```

```

    }
    case 7: {
        column_sort = "Disposed Value";
        break;
    }
}

if (column == 8) {
    return;
}

bool swapped;
MaterialRecord* current;
MaterialRecord* last = nullptr;

string direction_sort;
int direction = 1;
string print_direction_sort[14]= {"          Direction
Sort:", "Ascending", "Descending", "Back to Main Menu"};
int int_print_direction_sort = 4;
do {
    for(int j=0; j<int_print_direction_sort; j++) {
        if(j==direction) {
            SetConsoleTextAttribute(hConsole, 10);
            cout<<"          ->
"<<print_direction_sort[j]<<endl;
        } else
            cout<<"
"<<print_direction_sort[j]<<endl;
        SetConsoleTextAttribute(hConsole, 15);
    }
}

```



```

    }
    key=getch();
    if(key==KEY_S || key==KEY_DOWN) direction++;
    if(key==KEY_W || key==KEY_UP) direction--;
    if(direction<1) direction=1;
    if(direction>int_print_direction_sort-1)
direction=int_print_direction_sort-1;
    system("cls");
} while(key!=KEY_ENTER);

switch (direction) {
    case 1: {
        do {
            swapped = false;
            current = head;

            while (current->next != last) {
                if (compareByColumnUp(*current,
*(current->next), column)) {
                    swap(current, current->next);
                    swapped = true;
                }
                current = current->next;
            }
            last = current;
        } while (swapped);

        break;
    }
}

```

```

        case 2: {
            do {
                swapped = false;
                current = head;

                while (current->next != last) {
                    if (compareByColumnDown(*current,
*(current->next), column)) {
                        swap(current, current->next);
                        swapped = true;
                    }
                    current = current->next;
                }
                last = current;
            } while (swapped);
            break;
        }
        case 3: {
            return;
        }
    }

    system("cls");
    int choose = 1;
    string print_choose[3]= {"                      Records
sorted", "Back to Main Menu", "View Records"};
    int int_choose = 3;
    do {
        for(int j=0; j<int_choose; j++) {

```

```

        if(j==choose) {
            SetConsoleTextAttribute(hConsole, 10);
            cout<<" -> "<<print_choose[j]<<endl;
        } else
            cout<<"      "<<print_choose[j]<<endl;
        SetConsoleTextAttribute(hConsole, 15);
    }
    key=getch();
    if(key==KEY_S || key==KEY_DOWN) choose++;
    if(key==KEY_W || key==KEY_UP) choose--;
    if(choose<1) choose=1;
    if(choose>int_choose-1) choose=int_choose-1;
    system("cls");
} while(key!=KEY_ENTER);

switch (choose) {
    case 1: {
        break;
    }
    case 2: {
        displayRecords(head);
        system("cls");
        break;
    }
}

}

//#####clear
memory#####//

```

```

void releaseMemory(MaterialRecord*& head) {
    while (head != nullptr) {
        MaterialRecord* temp = head;
        head = head->next;
        delete temp;
    }
}

void exitProgram(MaterialRecord*& head) {
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    system("cls");
    int key;
    int choose = 1;
    string print_choose[4]= {"                Maybe save
Records?", "Yes", "No", "Back to Main Menu"};
    int int_choose = 4;
    do {
        for(int j=0; j<int_choose; j++) {
            if(j==choose) {
                SetConsoleTextAttribute(hConsole, 10);
                cout<<" -> "<<print_choose[j]<<endl;
            } else
                cout<<"      "<<print_choose[j]<<endl;
            SetConsoleTextAttribute(hConsole, 15);
        }
        key=getch();
        if(key==KEY_S || key==KEY_DOWN) choose++;
        if(key==KEY_W || key==KEY_UP) choose--;
    }
}

```



```

        if (key==KEY_S      ||      key==KEY_DOWN)
saving_choose++;
        if (key==KEY_W      ||      key==KEY_UP)
saving_choose--;
        if (saving_choose<1) saving_choose=1;
        if (saving_choose>int_saving_choose-1)
saving_choose=int_saving_choose-1;
        system("cls");
    } while (key!= KEY_ENTER);

    switch (saving_choose) {
        case 1: {
            string filename;
            cout << "Enter the name of the file
to save: ";

            cin >> filename;
            filename = filename + ".txt";
            saveToTextFile(head, filename);
            releaseMemory(head);
            system("cls");
            cout << "Exiting the program.
Memory has been freed." << endl;
            exit(0);
        }
        case 2: {
            saveToBinaryFile(head,
"material_records.bin");
            releaseMemory(head);

```

```

        cout << "Exiting the program.
Memory has been freed." << endl;
        exit(0);
    }
    case 3: {
        return;
    }
}
}
case 2: {
    releaseMemory(head);
    cout << "Exiting the program. Memory has
been freed." << endl;
    exit(0);
}
case 3: {
    return;
}
}
}

```

```

//#####poisk#
#####//

```

```

void searchByLastName(MaterialRecord* head, const
string& lastName) {
    MaterialRecord* current = head;
    bool found = false;
    if (lastName == "0") {

```

```

        system("cls");
        return;
    }

    while (current != nullptr) {
        if (current->LastName == lastName) {
            if (!found) {
                found = true;
                cout << "Search results for '" <<
lastName << "':" << endl;
                cout << "-----"
-----
-----" << endl;
                cout << "|    ID    | Factory | Branch |
Last Name   | Start Value | Received Value | Disposed
Value |" << endl;
                cout << "-----"
-----
-----" << endl;
            }
            printf("| %6d | %7d | %6d | %13s | %12.2f
| %14.2f | %14.2f |\n", current->id, current-
>factoryNumber,
                    current->branchNumber, current-
>LastName.c_str(), current->startValue,
                    current->receivedValue, current-
>disposedValue);
        }
        current = current->next;
    }

```



```

    }

    if (!found) {
        cout << "No records found for '" << lastName <<
        "'." << endl;
    } else {
        cout << "-----
-----
---" << endl;
    }
    cout << "-->Press any key to go back<--";
    getch();
    system("cls");
}

//#####report
#####//

void generateReport(MaterialRecord* head) {
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    int key, choice=1;
    string print_report[8] = { "          Choose
Report:",
                                "Calculate  End  Period
Value by Branch",
                                "Calculate  End  Period
Value by Factory",
                                "Calculate Totals",
                                "Print Report",

```

```

                                "Save      Report      to
'Report.txt'",
                                "Back to Main Menu",
                                "Exit"
                                };

    int int_print_report = 8;
    while (true) {
        do {
            for (int j = 0; j < int_print_report; j++)
            {
                if (j == choice) {
                    SetConsoleTextAttribute(hConsole,
10);

                    cout << " -> " << print_report[j]
<< endl;

                } else
                    cout << "      " << print_report[j]
<< endl;

                    SetConsoleTextAttribute(hConsole, 15);
                }
                key = getch();
                if(key==KEY_S || key==KEY_DOWN) choice++;
                if(key==KEY_W || key==KEY_UP) choice--;
                if (choice < 1) choice = 1;
                if (choice > int_print_report - 1) choice
= int_print_report - 1;
                    system("cls");
                } while (key != 13);

```

```

        switch (choice) {
            case 1: {
                cout << "\nEnd Period Value by Branch:"
<< endl;

                calculateEndPeriodValueByBranch(head);

                cout << "-->Press any key to go back<-
-";

                getch();
                system("cls");
                break;
            }
            case 2: {
                cout << "\nEnd Period Value by Branch:"
<< endl;

                calculateEndPeriodValueByFactory(head);

                cout << "-->Press any key to go back<-
-";

                getch();
                system("cls");
                break;
            }
            case 3: {
                cout << "Totals:" << endl;
                calculateTotals(head);
                cout << "-->Press any key to go back<-
-";

                getch();

```

```

        system("cls");
        break;
    }
    case 4: {
        printReport(head);
        cout << "-->Press any key to go back<-
-";

        getch();
        system("cls");
        break;
    }
    case 5: {
        printReportToFile(head);
        break;
    }
    case 6: {
        return;
    }
    case 7: {
        exitProgram(head);
    }
}

}
}

```

```

void printReport(MaterialRecord* head) {
    map<int, double> totalStartValueByFactory;
    map<int, double> totalReceivedValueByFactory;
    map<int, double> totalDisposedValueByFactory;

```

```

    map<int, double> totalEndValueByFactory;

    map<pair<int,          int>,          double>
totalStartValueByBranch;
    map<pair<int,          int>,          double>
totalReceivedValueByBranch;
    map<pair<int,          int>,          double>
totalDisposedValueByBranch;
    map<pair<int, int>, double> totalEndValueByBranch;

    double totalStartValue = 0.0;
    double totalReceivedValue = 0.0;
    double totalDisposedValue = 0.0;
    double totalEndValue = 0.0;

    MaterialRecord* current = head;

    while (current != nullptr) {
        pair<int,          int>          branchFactoryPair          =
make_pair(current->branchNumber,          current->
>factoryNumber);

        if
(totalEndValueByBranch.find(branchFactoryPair)          !=
totalEndValueByBranch.end()) {
            totalEndValueByBranch[branchFactoryPair]
+=  current->startValue  +  current->receivedValue  -
current->disposedValue;
        } else {

```

```

        totalEndValueByBranch[branchFactoryPair] =
current->startValue + current->receivedValue - current-
>disposedValue;
    }

    if
(totalDisposedValueByBranch.find(branchFactoryPair) !=
totalDisposedValueByBranch.end()) {

        totalDisposedValueByBranch[branchFactoryPair] +=
current->disposedValue;
    } else {

        totalDisposedValueByBranch[branchFactoryPair] =
current->disposedValue;
    }

    if
(totalReceivedValueByBranch.find(branchFactoryPair) !=
totalReceivedValueByBranch.end()) {

        totalReceivedValueByBranch[branchFactoryPair] +=
current->receivedValue;
    } else {

        totalReceivedValueByBranch[branchFactoryPair] =
current->receivedValue;
    }

```

```

        if
        (totalStartValueByBranch.find(branchFactoryPair)      !=
totalStartValueByBranch.end()) {
            totalStartValueByBranch[branchFactoryPair]
+= current->startValue;
        } else {
            totalStartValueByBranch[branchFactoryPair]
= current->startValue;
        }

        totalStartValueByFactory[current-
>factoryNumber] += current->startValue;
        totalReceivedValueByFactory[current-
>factoryNumber] += current->receivedValue;
        totalDisposedValueByFactory[current-
>factoryNumber] += current->disposedValue;

        totalStartValue += current->startValue;
        totalReceivedValue += current->receivedValue;
        totalDisposedValue += current->disposedValue;

        current = current->next;
    }

    // Вывод данных по каждому филиалу
    cout << "                                REPORT"
<< endl;
    cout << "-----"
-----" << endl;

```

```

        cout << "| Factory | Branch | Start Value |  

Received Value | Disposed Value | End Value |" <<  

endl;  

        cout << "-----  

-----" << endl;  

        cout << fixed << setprecision(2);  

        for (const auto& pairValue : totalEndValueByBranch)  

{  

            cout << " | " << setw(7) <<  

pairValue.first.second << " | " << setw(6) <<  

pairValue.first.first << " | "  

<< setw(13) <<  

totalStartValueByBranch[pairValue.first] << " | " <<  

setw(14) << totalReceivedValueByBranch[pairValue.first]  

<< " | " << setw(14) <<  

totalDisposedValueByBranch[pairValue.first] << " | " <<  

setw(13) << pairValue.second << " |" << endl;  

        }  

        // Вывод данных по каждому заводу  

        cout << "  

=====
```



```

        cout << "-----
-----" << endl;

        cout << fixed << setprecision(2);
        for (const auto& factory : totalStartValueByFactory)
        {
            totalEndValueByFactory[factory.first] =
factory.second +
totalReceivedValueByFactory[factory.first] -
totalDisposedValueByFactory[factory.first];
            cout << "| " << setw(16) << factory.first << "
| " << setw(13) << factory.second << " | " << setw(14)
<<
totalReceivedValueByFactory[factory.first] << " | " <<
setw(14) << totalDisposedValueByFactory[factory.first]
<< " | " << setw(13) <<
totalEndValueByFactory[factory.first] << " |" << endl;
        }

        // Вывод суммарных данных по всем заводам
        totalEndValue = totalStartValue +
totalReceivedValue - totalDisposedValue;
        cout << "-----
-----" << endl;

        cout << "| Total: | " << setw(13) <<
totalStartValue << " | " << setw(14) <<
totalReceivedValue << " | "
<< setw(14) << totalDisposedValue << " | " <<
setw(13) << totalEndValue << " |" << endl;

```

```

        cout << "-----
-----" << endl;
    }

void printReportToFile(MaterialRecord* head) {
    ofstream file("Report.txt");
    if (!file.is_open()) {
        cout << "Unable to open file!" << endl;
        return;
    }

    streambuf* original = cout.rdbuf(file.rdbuf());
    printReport(head);
    cout.rdbuf(original);
    file.close();

    cout << "Report saved to text file successfully!"
<< endl;
    cout << "-->Press any key to go back<--";
    getch();
    system("cls");
}

void calculateEndPeriodValueByBranch(MaterialRecord*
head) {
    map<pair<int, int>, double> endPeriodValueByBranch;

    MaterialRecord* current = head;

```

```

        while (current != nullptr) {
            pair<int, int> branchFactoryPair =
make_pair(current->branchNumber, current->factoryNumber);

            if
(endPeriodValueByBranch.find(branchFactoryPair) !=
endPeriodValueByBranch.end()) {
                endPeriodValueByBranch[branchFactoryPair]
+= current->startValue + current->receivedValue -
current->disposedValue;
            } else {
                endPeriodValueByBranch[branchFactoryPair]
= current->startValue + current->receivedValue -
current->disposedValue;
            }
            current = current->next;
        }

        cout << "-----" <<
endl;
        cout << "| Factory | Branch | End Period Value |"
<< endl;
        cout << "-----" <<
endl;

        cout << fixed << setprecision(2);

```

```

        for (const auto& pairValue :
endPeriodValueByBranch) {
            cout << " | " << setw(7) <<
pairValue.first.second << " | " << setw(6) <<
pairValue.first.first << " | " << setw(16) <<
pairValue.second << " |" << endl;
        }

        cout << "-----" <<
endl;
        return;
    }

void calculateEndPeriodValueByFactory(MaterialRecord*
head) {
    map<int, double> endPeriodValueByFactory;

    MaterialRecord* current = head;

    while (current != nullptr) {
        if (endPeriodValueByFactory.find(current-
>factoryNumber) != endPeriodValueByFactory.end()) {
            endPeriodValueByFactory[current-
>factoryNumber] += current->startValue + current-
>receivedValue - current->disposedValue;
        } else {
            endPeriodValueByFactory[current-
>factoryNumber] = current->startValue + current-
>receivedValue - current->disposedValue;

```

```

        }
        current = current->next;
    }

    cout << "-----" << endl;
    cout << "| Factory | End Period Value |" << endl;
    cout << "-----" << endl;

    cout << fixed << setprecision(2);

    for (const auto& pairValue :
endPeriodValueByFactory) {
        cout << "| " << setw(7) << pairValue.first <<
" | " << setw(16) << pairValue.second << " |" << endl;
    }

    cout << "-----" << endl;
    return;
}

void calculateTotals(MaterialRecord* head) {
    double startValueTotal = 0.0;
    double receivedValueTotal = 0.0;
    double disposedValueTotal = 0.0;
    double endPeriodValueTotal = 0.0;

    MaterialRecord* current = head;

```

```

while (current != nullptr) {
    startValueTotal += current->startValue;
    receivedValueTotal += current->receivedValue;
    disposedValueTotal += current->disposedValue;
    endPeriodValueTotal += (current->startValue +
current->receivedValue - current->disposedValue);

    current = current->next;
}

cout << "-----" << endl;
cout << "| Type          | Total Value |" << endl;
cout << "-----" << endl;
cout << "| Start Value | " << setw(11) <<
startValueTotal << " |" << endl;
cout << "| Received      | " << setw(11) <<
receivedValueTotal << " |" << endl;
cout << "| Disposed      | " << setw(11) <<
disposedValueTotal << " |" << endl;
cout << "| End Period    | " << setw(11) <<
endPeriodValueTotal << " |" << endl;
cout << "-----" << endl;
return;
}

```

```

int main() {
    MaterialRecord* records = nullptr;

```

```

int key,choice=1;
HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
string print_main[13]= {"Menu:", "View
Records", "Add Record", "Delete Record", "Edit Record",
                        "Save Records to
file.txt", "Save Records to file.bin", "Load Records from
file",
                        "Sort Records", "Search by
LastName", "Clear Records", "Generate Report", "Exit"
                        };
int int_print_main = 13;
while(true) {
    do {
        for(int j=0; j<int_print_main; j++) {
            if(j==choice) {
                SetConsoleTextAttribute(hConsole,
10);

                cout<<" -> "<<print_main[j]<<endl;
            } else
                cout<<" "<<print_main[j]<<endl;
            SetConsoleTextAttribute(hConsole, 15);
        }
        key=getch();
        if(key==KEY_S || key==KEY_DOWN) choice++;
        if(key==KEY_W || key==KEY_UP) choice--;
        if(choice<1) choice=1;
        if(choice>int_print_main-1)
choice=int_print_main-1;

```

```

        system("cls");
    } while (key!= KEY_ENTER);

    switch (choice) {
        case 1: {
            cout << "\nMaterial Records:" << endl;
            displayRecords(records);
            system("cls");
            break;
        }
        case 2: {
            int factory, branch;
            string lastName;
            double start, received, disposed;

            cout << "Enter 'esc' to go to the main
menu, enter any key to continue"<<endl;
            char key = _getch();
            if (key == KEY_ESC) {
                system("cls");
                break;
            }
            system("cls");
            cout << "Enter Factory Number: ";
            cin >> factory;
            cout << "Enter Branch Number: ";
            cin >> branch;
            cout << "Enter Last Name: ";

```



```

        cin >> lastName;
        cout << "Enter Start Value: ";
        cin >> start;
        cout << "Enter Received Value: ";
        cin >> received;
        cout << "Enter Disposed Value: ";
        cin >> disposed;

        addRecord(records,
createRecord(factory,    branch,    lastName,    start,
received, disposed));

        cout << "-->Press any key to go back<-
-";

        getch();
        system("cls");
        break;
    }
    case 3: {
        int idToDelete;
        cout << "Enter '0' to go to the main
menu"<<endl<<"Enter ID of the record to delete: ";
        cin >> idToDelete;
        system("cls");
        deleteRecord(records, idToDelete);
        break;
    }
    case 4: {
        int idToEdit;

```

```

        cout << "Enter '0' to go to the main
menu"<<endl<<"Enter ID of the record to edit: ";
        cin >> idToEdit;
        system("cls");
        editRecord(records, idToEdit);
        break;
    }
    case 5: {
        string filename;
        cout << "Enter the name of the file to
save: ";

        cin >> filename;
        filename = filename + ".txt";
        system("cls");
        saveToTextFile(records, filename);
        break;
    }
    case 6: {
        system("cls");
        saveToBinaryFile(records,
"Material_Records.bin");
        break;
    }
    case 7: {
        string filename;
        cout << "Enter the name of the file to
save: ";

        cin >> filename;
        filename = filename + ".txt";

```

```

        system("cls");
        loadFromTextFile(records, filename);
        break;
    }
    case 8: {
        sortRecords(records);
        break;
    }
    case 9: {
        string searchLastName;
        cout << "Enter '0' to go to the main
menu"<<endl<< "Enter Last Name to search: ";
        cin >> searchLastName;
        searchByLastName(records,
searchLastName);
        break;
    }
    case 10: {
        releaseMemory(records);
        cout << "Records has been cleared."
<<endl;

        cout << "-->Press any key to go back<-
-";

        getch();
        system("cls");
        break;
    }
    case 11: {
        generateReport(records);

```

```
        break;
    }
    case 12: {
        exitProgram(records);
    }
}
}
```