Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра вычислительных методов и программирования

Дисциплина: Основы алгоритмизации и программирования

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**
к курсовой работе
на тему:

# «СИСТЕМА РАСЧЕТА СТИПЕНДИИ ОБУЧАЮЩИМСЯ»

Выполнил: гр.421702 Захаренков И.Д.

Проверил: Панасик А.А.

Минск 2025

# РЕФЕРАТ

СИСТЕМА РАСЧЕТА СТИПЕНДИИ ОБУЧАЮЩИМСЯ: курсовая работа / Захаренков И.Д. – Минск: БГУИР, 2025.

В данной работе рассматривается разработка программного средства для автоматизации системы расчета стипендии обучающихся. Основная цель проекта – создание удобной и эффективной программы, которая позволяет выполнять основные операции по обработке бинарных файлов в консольном режиме, существенно облегчая работу и анализ данных студентов.

Проект реализует стандартные операции обработки файлов данных: создание нового файла, просмотр, добавление, редактирование и удаление записей. Также реализованы алгоритмы линейного поиска валют по ФИО и бинарного поиска по среднему баллу, а также алгоритмы сортировки: быстрая сортировка по стипендии, сортировка выбором по среднему баллу и сортировка вставками по ФИО. Дополнительно предусмотрена функция создания отчётов в текстовом файле и проведение статистики по студентам для увеличения стипендии и статистики по стипендиям.

Курсовая работа включает текст и иллюстрации, наглядно раскрывающие проектные решения. Пояснительная записка содержит подробное описание используемых структур данных, алгоритмов сортировки и поиска, пользовательских функций, интерфейса и общей логики функционирования программного продукта

# СОДЕРЖАНИЕ

Введение.....6
1 Структуры и файлы.....7
1.1 Структура Student.....7
1.2 Работа с файлами.....8
2 Алгоритмы сортировки.....10
2.1 Быстрая сортировка (по размеру стипендии).....10
2.2 Сортировка выбором (по среднему баллу).....11
2.3 Сортировка вставками (по ФИО).....12
3 Алгоритмы поиска.....14
3.1 Линейный поиск (по ФИО).....14
3.2 Бинарный поиск (по среднему баллу).....15
4 Пользовательские функции.....17
4.1 Вспомогательные функции.....17
4.2 Функции работы с базой данных.....18
4.3 Функции поиска и анализа данных.....20
4.4 Управление памятью и файлами.....22
5 Описание работы программы.....24
5.1 Интерфейс и основные функции.....24
5.2 Работа с базой данных.....24
5.3 Поиск данных.....26
5.4 Сортировка.....26
5.5 Статистический анализ.....27
5.6 Пользовательский опыт.....28
Заключение.....29
Список использованных источников.....30
Приложение А (обязательное) Листинг кода.....31
Приложение Б (обязательное) Блок-схема работы программы.....63

# ВВЕДЕНИЕ

В наши дни информационные технологии играют важную роль в автоматизации процессов управления данными. Одной из важных задач в этой области является разработка программного обеспечения, позволяющего эффективно хранить, обрабатывать и анализировать информацию. В условиях роста объемов данных все большее значение приобретают алгоритмы сортировки и поиска, которые обеспечивают быстрое извлечение информации, ее упорядочение и последующее использование. Данные методы нашли широкое применение в управлении базами данных.

Объектом исследования в данной работе выступает система управления учебными данными студентов, реализованная в виде консольного приложения на языке C++. Программа предоставляет возможность вести базу данных студентов, выполнять поиск и сортировку записей, а также проводить статистический анализ информации. Основное внимание уделено применению различных алгоритмов сортировки и поиска для повышения эффективности работы с данными.

Целью курсовой работы стало создание функционального консольного приложения, позволяющего осуществлять операции над базой данных студентов, включая добавление, удаление, редактирование, сортировку, поиск и формирование отчетов.

Результатом работы стал прототип системы, который может быть адаптирован для использования в образовательных учреждениях, а также использован как основа для дальнейшего развития в направлении графического интерфейса и интеграции с внешними СУБД.

# 1 СТРУКТУРЫ И ФАЙЛЫ

В рамках реализации программы для управления базой данных студентов были разработаны структуры данных и механизмы работы с файлами, обеспечивающие хранение, обработку и извлечение информации [9]. Основной структурой данных выступает структура `Student`, который инкапсулирует данные о студенте и предоставляет методы для их манипуляции. Для сохранения данных между сессиями программы используется бинарный файл `Storage.bin`, поддерживающий сериализацию и десериализацию массива объектов `Student`. Дополнительно для вывода результатов анализа данных (статистика стипендий, результаты поиска) применяется текстовый файл `result.txt`.

## 1.1 Структура Student

Структура Student предназначена для хранения информации о студенте, включая его ФИО, факультет, средний балл, наличие льгот и размер стипендии.

Каждое поле структуры имеет тип, выбранный с учетом специфики данных:

– `fullName` (тип char*) – хранит ФИО студента. Использование указателя на строку позволяет свободно управлять текстовой информацией переменной длины, обеспечивая возможность ввода и хранения ФИО разных размеров;

– `faculty` (тип char*) – хранит название факультета, на котором обучается студент. Как и `fullName`, реализован как указатель на строку для адаптации к произвольной длине наименований факультетов;

– `averageGrade` (тип double) – хранит средний балл студента. Тип double выбран для точного представления числа с плавающей точкой, что помогает увеличить точность на математических рассчетах;

– `hasPrivileges` (тип bool) – флаг, указывающий на наличие у студента льгот (например, социальных или академических). Логический тип (true/false) обеспечивает простое и однозначное представление двоичного состояния;

– `scholarshipAmount` (тип double) – отражает размер стипендии студента. Как и `averageGrade`, реализован через double для точных вычислений и хранения дробных значений.

Для корректного управления динамической памятью реализованы конструктор копирования и перегруженный оператор присвоения. Они обеспечивают глубокое копирование строк, что помогает избежать проблем с копированием указателей и делает работу с объектами безопаснее.

Пример объявления структуры:

```
struct Student {
private:
```

```
    char *fullName;
    char *faculty;
    double averageGrade;
    bool hasPrivileges;
    double scholarshipAmount;
public:
    // Конструктор, геттеры, сеттеры и другие методы
};
```

Также предусмотрены методы вывода данных в консоль и файл, что упрощает отладку и экспорт информации.

### 1.2 Работа с файлами

Для долгосрочного хранения данных используется бинарный файл Storage.bin, в котором информация о студентах хранится в компактном виде.

Формат записи включает:

– Запись количества студентов (`size_t size`) в начало файла;

– Последовательную запись каждого студента, включая длины строковых полей (`fullName, faculty`) и их содержимое, а также числовые и булевые значения (`averageGrade, hasPrivileges, scholarshipAmount`).

Методы `FileManager::readStudentsFromFile` и `FileManager::writeStudentsToFile` обеспечивают чтение и запись данных, корректно обрабатывая динамические массивы и предотвращая утечки памяти. Для повышения эффективности при чтении данных используется функция `ResizeArray`, которая выделяет память под массив студентов с запасом, основываясь на степень двойки. Это снижает частоту перераспределения памяти при добавлении новых записей.

Пример фрагмента кода для чтения данных из файла:

```
bool FileManager::readStudentsFromFile(const char *filename,
Student **&students, size_t &size) {
    std::ifstream file(filename, std::ifstream::in);
    file.read(reinterpret_cast<char*>(&size),
sizeof(size_t));
    // Чтение данных о каждом студенте
}
```

Для вывода результатов анализа данных (например, статистику по стипендиям или результаты поиска) используется текстовый файл result.txt. Функции `writeScholarshipStatisticsToFile` и `writeIncreasedScholarshipStatisticsToFile` создают отчеты, которые удобно анализировать. В статистике по стипендиям есть минимальные и максимальные значения для каждого факультета, а также список студентов, которые соответствуют заданным критериям.

Пример записи статистики в result.txt:

```
    bool FileManager::writeScholarshipStatisticsToFile(const
char *filename, Student **students,
        size_t size, char* faculty, double minS, double maxS) {
        std::ofstream file(filename, std::ofstream::app);
        file << "----- " << faculty << " -----" << std::endl;
        file << "Min scholarship: " << minS << std::endl;
        file << "Max scholarship: " << maxS << std::endl;
        // Запись данных о студентах
    }
```

Этот способ работы с файлами помогает быстро обрабатывать большие объемы данных и подходит для разных операционных систем, так как использует стандартные потоки ввода–вывода C++.

# 2 АЛГОРИТМЫ СОРТИРОВКИ

Алгоритмы сортировки являются одной из фундаментальных областей изучения в информатике [1]. Их применение оправдано в задачах, где требуется упорядочить данные для последующего анализа, поиска или использования в алгоритмах. В контексте разработанной программы сортировка позволяет структурировать информацию о студентах по заданным критериям, таким как размер стипендии, средний балл или алфавитный порядок в ФИО. Это упрощает работу с базой данных и делает поиск и фильтрацию более удобными, а также помогает пользователю лучше видеть результаты.

Выбор конкретного алгоритма зависит от множества факторов: объема данных, характера входных данных (например, частично упорядоченные или случайные массивы), требований к скорости выполнения и используемой памяти. В данной работе реализованы три алгоритма – быстрая сортировка, сортировка выбором и сортировка вставками. Каждый из алгоритмов имеет свои сильные и слабые стороны, что делает их применимыми в разных условиях. Например, быстрая сортировка показывает высокую производительность на больших наборах данных, тогда как сортировка вставками оказывается более эффективной при небольших объемах данных или почти отсортированных массивах. Сортировка выбором, в свою очередь, проста в реализации и минимизирует количество перестановок, что может быть важно в некоторых ситуациях.

Все три алгоритма адаптированы для работы с массивами указателей на объекты структуры `Student`, что позволяет сохранять целостность данных при сортировке и не требует делать копии объектов. Это особенно полезно при больших объемах информации, так как уменьшает нагрузку на память и на вычислительные процессы. Дальнейшее описание каждого алгоритма (см. разделы 2.1–2.3) включает принципы их работы, особенности реализации в рамках проекта и причинах выбора для конкретных задач.

## 2.1 Быстрая сортировка (по размеру стипендии)

Быстрая сортировка (или же quick sort) – это один из популярных алгоритмов для сортировки, который действует по принципу «разделяй и властвуй» [1]. Этот метод подходит для разных задач, особенно там, где нужно быстро обрабатывать большие объемы данных, например, в базах данных и в машинном обучении. В данной программе он реализован для сортировки массива студентов по возрастанию размера стипендии.

Алгоритм работает следующим образом (пример работы алгоритма показан в приложении А):

1 Выбор опорного элемента. В качестве опорного выбирается последний элемент массива (например, `arr[high]`, где high – индекс последнего элемента);

2 Разделение массива. Мы переставляем элементы так, чтобы все, что меньше опорного, оказалось слева, а всё, что больше – справа. Для этого используется вспомогательная функция `QuickSortPartition`, которая возвращает индекс опорного элемента после разделения;

3 Рекурсивная сортировка подмассивов. Процесс повторяется рекурсивно для всех подмассивов, расположенных слева и справа от опорного элемента.

Алгоритм адаптирован для работы с массивами указателей на объекты структуры `Student`, что помогает снизить вычислительные затраты на копирование данных. Вместо перемещения самих объектов производится обмен указателями, благодаря чему повышается скорость обработки больших объемов данных.

Преимущества быстрой сортировки:

– Высокая скорость работы. В среднем его сложность составляет O(nlog(n)), что делает его хорошим выбором для больших наборов.

– Минимизация использования дополнительной памяти. Сортировка происходит «на месте», без создания временных массивов;

– Гибкость. Алгоритм легко адаптируется под разные критерии сравнения (в данном случае – сравнение значений поля `scholarshipAmount`).

Важно помнить, что эффективность быстрой сортировки напрямую зависит от выбора опорного элемента. В худшем случае, когда массив уже отсортирован или состоит из одинаковых элементов, временная сложность алгоритма может упасть до $O(n^2)$. Во избежание такого сценария в реальных проектах часто применяются модификации, такие как случайный выбор опорного элемента или использование трехстороннего разделения. В данной работе выбор последнего элемента в качестве опорного сделан из–за простоты реализации и достаточной производительности.

Для обратной сортировки (по убыванию стипендии) реализована аналогичная функция `QuickSortByScholarshipReverse`, где условие сравнения элементов инвертировано.

### 2.2 Сортировка выбором (по среднему баллу)

Сортировка выбором (или же selection sort) – это простой, но эффективный алгоритм сортировки, основанный на идее последовательного поиска минимального или максимального элемента в неотсортированной части массива с последующим его перемещением на соответствующую позицию [1]. В разработанной программе этот алгоритм адаптирован для сортировки студентов по возрастанию среднего балла. Выбор данного алгоритма обусловлен минимальным количеством перестановок элементов, что может быть актуально при работе с данными, которые хранятся в файлах и требуют аккуратного управления памятью.

Принцип работы сортировки выбором заключается в следующем (пример работы алгоритма показан в приложении А):

1 Итерация по массиву. проходим по массиву и на каждой итерации находим студента с самым низким средним баллом в оставшейся неотсортированной части.

2 Обмен элементов. Найденный элемент перемещается в начало неотсортированной части массива, меняясь местами с элементом на первой позиции.

3 Сужение диапазона. После каждой итерации граница между отсортированной и неотсортированной частями сдвигается вправо, уменьшая объем работы на следующем шаге.

Алгоритм работает с массивами указателей на объекты структуры `Student`, что помогает избежать лишних затрат на копирование данных. Вместо перемещения самих объектов производится обмен адресами, что делает процесс быстрее, особенно с большим объемом данных. Для сортировки по убыванию баллов у нас есть функция `SelectionSortByGradeReverse`, где мы просто изменяем условие сравнения.

Преимущества сортировки выбором:

– Минимальное количество перестановок. Алгоритм выполняет всего O(n) обменов, что делает его предпочтительным при работе с ресурсами, где операции записи требуют значительных затрат;

– Простота реализации. Код легко читать и модифицировать, что облегчает будущие изменения;

– Стабильность на частично упорядоченных данных. Он сохраняет свою эффективность, даже если часть данных уже отсортирована.

Однако временная сложность сортировки выбором составляет $O(n^2)$ во всех случаях, что делает его менее подходящим для больших объемов данных по сравнению с быстрой сортировкой. Тем не менее, в сценариях, где требуется сортировка небольшого числа записей (например, анализ успеваемости на конкретном факультете), этот метод вполне приемлем по скорости и надежности.

### 2.3 Сортировка вставками (по ФИО)

Сортировка вставками (или же insertion sort) – это простой, но достаточно эффективный алгоритм сортировки [1]. В данной программе он используется для упорядочивания массива студентов по алфавиту, основываясь на ФИО. Алгоритм эффективен при работе с небольшими или частично отсортированными наборами данных, поэтому он подходит для программ с интерактивным интерфейсом.

Принцип работы сортировки заключается в последовательном добавлении элементов в отсортированную часть массива (пример работы алгоритма находится в приложении А):

1 Итерация по массиву. Начинаем со второго элемента, который рассматриваем как кандидата для вставки (key) в отсортированную (левую) часть массива;

2 Сравнение и сдвиг. Элементы из отсортированной части массива последовательно сравниваются с key. Если элемент больше key, он сдвигается вправо, освобождая место для вставки;

3 Вставка. Когда подходящая позиция найдена, вставляем key в массив.

Алгоритм адаптирован для работы с массивами указателей на объекты структуры `Student`, что помогает избежать лишних затрат на копирование данных. Вместо перемещения самих объектов производится обмен адресами, что значительно ускоряет процесс, особенно при работе с текстовыми данными переменной длины. Для сортировки в обратном порядке (по убыванию) есть функция `InsertionSortByNameReverse`, где условия сравнения перевернуты.

Преимущества сортировки вставками:

– Простота реализации. Код легко читается и изменяется, что упрощает дальнейшие изменения;

– Эффективность на малых и почти отсортированных данных. В лучшем случае, когда массив уже отсортирован, временная сложность составляет O(n), так что данный алгоритм хорошо подходит для динамических данных;

– Стабильность. Порядок одинаковых элементов сохраняется, что важно, например, при сортировке студентов с одинаковыми фамилиями.

Однако временная сложность в среднем и худшем случае составляет $O(n^2)$, что делает сортировку вставками менее эффективным для больших объемов данных по сравнению с быстрой сортировкой. Тем не менее, в сценариях, где требуется интерактивная сортировка по ФИО (например, поиск студента по имени или при форматировании массива), этот метод демонстрирует достаточную скорость и надежность.

# 3 АЛГОРИТМЫ ПОИСКА

Алгоритмы поиска играю важную роль в информационных системах, обеспечивая возможность быстрого и эффективного извлечения данных из хранилищ информации. Их применение оправдано в задачах, где требуется находить элементы, соответствующие заданным критериям, среди большого объема данных [5]. В контексте разработанной программы поиск позволяет быстро находить студентов по имени или среднему баллу в базе данных, фильтровать записи или формировать отчеты. Например, при поиске студента по имени алгоритм позволяет за минимальное время получить список кандидатов, а бинарный поиск по среднему баллу обеспечивает быстрое извлечение данных для анализа успеваемости.

Выбор алгоритма зависит от множества факторов: объема данных, типа входных параметров (например, необходимость работы с неотсортированными или частично упорядоченными массивами), требований к скорости выполнения или объему памяти. В этой работе использованы два алгоритма – линейный и бинарный поиск. Каждый из них имеет свои плюсы и минусы, что делает их применимыми в различных сценариях использования. Линейный поиск прост в реализации и может работать с любыми данными, включая неотсортированные массивы, но демонстрирует линейную сложность O(n), что делает медленным на больших объемах данных. Бинарный поиск, напротив, требует предварительной сортировки данных, но обеспечивает логарифмическую сложность O(log(n)), что может быть важно при работе с крупными наборами данных.

Оба алгоритма адаптированы для работы с массивами указателей на объекты структуры `Student`, что позволяет сохранять целостность данных при поиске без необходимости копирования самих объектов. Это особенно важно при работе с большими объемами информации, так как снижает затраты на перемещение данных в памяти. Далее (в разделах 3.1–3.2) будут описаны принципы работы каждого алгоритма, особенности реализации и причины выбора для конкретных задач.

## 3.1 Линейный поиск (по ФИО)

Линейный поиск (или же linear search) представляет собой один из наиболее простых и интуитивно понятных алгоритмов поиска данных в массиве. В разработанной программе он адаптирован для поиска студентов по полному совпадению ФИО. Алгоритм последовательно просматривает каждый элемент массива, сравнивая его с нужным значением, и останавливается при совпадении или достижении конца массива. Такой подход не требует предварительной сортировки данных, что позволяет работать с любыми наборами данных.

Принцип работы линейного поиска заключается в следующем (пример работы алгоритма находится в приложении А):

1 Инициализация. На вход подается массив указателей на объекты Student, размер массива и целевое значение (name – ФИО для поиска);

2 Итерация по массиву. Для каждого элемента массива выполняется сравнение строки `fullName` текущего студента с целевым значением с помощью функции strcmp;

3 Формирование результата. При совпадении указатель на соответствующий объект добавляется в выходной массив, а счетчик найденных элементов (`outputSize`) увеличивается.

Алгоритм адаптирован для работы с массивами указателей на объекты структуры `Student`, что позволяет сохранять целостность данных при поиске без необходимости копирования самих объектов. Это особенно важно при работе с большими объемами информации, так как экономит ресурсы.

Преимущества линейного поиска:

– Простота реализации. Алгоритм легко читается и модифицируется, что упрощает его интеграцию в программу;

– Независимость от упорядоченности данных. Он подходит как для отсортированных, так и для неотсортированных массивов;

– Минимальные требования к памяти. Не требует создания дополнительных структур данных, кроме выходного массива.

Однако временная сложность линейного поиска составляет O(n), что делает его не эффективным на больших объемах данных по сравнению с тем же бинарным поиском [5]. Тем не менее, когда нужно точно найти студента по ФИО, этот метод работает быстро и надежно.

### 3.2 Бинарный поиск (по среднему баллу)

Бинарный поиск (или же binary search) – это достаточно простой и быстрый алгоритм поиска элемента в отсортированном массиве, основанный на принципе последовательного деления диапазона поиска пополам [5]. В разработанной программе он адаптирован для поиска студентов по значению среднего балла. Для корректной работы алгоритма массив предварительно сортируется по возрастанию среднего балла с помощью `SelectionSortByGrade`.

Принцип работы бинарного поиска включает следующие шаги (пример работы алгоритма показан в приложении А):

1 Инициализация. На вход подается отсортированный массив указателей на объекты `Student`, размер массива и целевое значение (grade – средний балл для поиска);

2 Поиск левой границы. Алгоритм определяет первый элемент массива, значение которого равно или больше целевого. Для этого диапазон поиска последовательно сужается путем сравнения среднего элемента (mid) с целевым значением;

3 Сбор результатов. В конце, когда мы находим эту границу, все последующие элементы с тем же значением добавляются в результат.

Алгоритм адаптирован для работы с массивами указателей на объекты структуры `Student`. Это позволяет сохранять целостность данных при поиске и избегать лишних копирований самих объектов. Особенно это важно при работе с большими объёмами информации – такой подход уменьшает нагрузку на память и повышает производительность.

Преимущества бинарного поиска:

– Высокая скорость работы. Временная сложность алгоритма составляет O(log(n)), что делает его отличным выбором для обработки больших наборов данных;

– Минимизация сравнений. На каждом шаге область поиска сокращается вдвое, благодаря чему количество операций остаётся минимальным;

– Возможность находить несколько совпадений. Алгоритм может не только найти первый попавшийся элемент с нужным значением среднего балла, но и выявить все такие элементы.

Однако бинарный поиск имеет ограничения:

– Необходимость предварительной сортировки. Если массив не упорядочен, алгоритм может вернуть неверный результат или вообще не найти нужный элемент;

– Не подходит для часто изменяемых данных. При частых добавлениях или удалениях элементов массив приходится постоянно сортировать, что снижает общую эффективность.4 Пользовательские функции

# 4 ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ

При разработке программы были созданы собственные функции, которые отвечают за работу с данными студентов – включая их хранение, обработку и выполнение специфических операций. Эти функции берут на себя управление памятью, взаимодействие с данными и реализацию ключевых логических процессов. Они позволяют структурировать код, упрощают его сопровождение и делают программу более гибкой и надёжной.

## 4.1 Вспомогательные функции

В рамках реализации программы были разработаны вспомогательные функции, обеспечивающие выполнение специфических операций, необходимых для корректной работы с данными студентов, управления памятью и обработки пользовательского ввода. Эти функции не связаны напрямую с основными алгоритмами сортировки или поиска, но при этом играют важную роль в обеспечении стабильной и корректной работы всей программы.

Одной из задач программы является группировка студентов по факультетам, необходимая для анализа стипендий и формирования статистики. Функция `findFaculties` решает проблему определения уникальных факультетов среди всех записей в массиве студентов. Для этого она проходит по каждому элементу массива, сравнивая текущий факультет с уже найденными значениями. Если факультет встречается впервые, он добавляется в выходной список. Такой подход позволяет избежать дублирования данных и обеспечивает корректную группировку при последующей обработке. Например, при формировании отчета о стипендиях по факультетам функция гарантирует, что каждый факультет будет обработан только один раз, что делает программу более эффективной.

Пользовательский ввод таких данных, как средний балл или размер стипендии, требует обязательной проверки – это важно для предотвращения ошибок выполнения и обеспечения корректности данных.

Для решения этой задачи используется функция `isDigit`, которая проверяет, является ли введённая пользователем строка допустимым числовым значением [3]. Функция анализирует каждый символ строки, убеждаясь, что он соответствует цифре, а также проверяет правильность использования разделителя – точки или запятой – в случае дробных чисел.

Работа с динамической памятью в C++ требует особого внимания, так как некорректное управление ресурсами может привести к утечкам памяти или неопределенному поведению программы. Так, функция `deleteStudentsArray` реализует безопасное освобождение памяти, выделенной под массив студентов. Она последовательно удаляет объекты `Student`, а затем сам массив указателей, предварительно проверяя корректность входных параметров. Эта функция применяется во всех частях программы, где завершается работа с массивом студентов – например, после

чтения данных из файла или обработки результатов поиска. Её использование обеспечивает своевременное освобождение памяти, занятой временными данными, что особенно важно при частом выполнении операций чтения и записи, а также при работе с большими объёмами информации.

Сортировка студентов по алфавиту требует точное и корректное сравнение строк, учитывающее особенности лексикографического порядка. Функция `compareChars` реализует механизм сравнения, похожий на стандартную функцию `strcmp`, но адаптированный для работы с динамическими строками, хранящимися в структуре `Student`. Она возвращает логическое значение, указывающее, какая из строк должна идти первой – это позволяет ее использовать в качестве критерия в алгоритме сортировки вставками. Функция учитывает регистр символов и длину строк, обеспечивая стабильный и предсказуемый результат даже при наличии сложных комбинаций букв. Это особенно важно при выводе данных в отсортированном виде, когда пользователь ожидает, что записи будут расположены логично – например, в строгом алфавитном порядке.

### 4.2 Функции работы с базой данных

Функции работы с базой данных обеспечивают взаимодействие программы с хранилищем данных, которое реализовано в виде бинарного файла Storage.bin. Эти функции отвечают за выполнение операции чтения, добавления, удаления, редактирования и анализа записей о студентах, обеспечивая целостность данных. Их реализация учитывает особенности работы с динамической памятью, преобразования данных и обработку исключений, связанных с вводом–выводом.

Функция `displayAll` предназначена для вывода информации обо всех студентах, которые хранятся в базе данных. Она считывает данные из файла Storage.bin с помощью `readStudentsFromFile`, форматирует их для отображения в консоли и после завершения работы освобождает выделенную под данные память. Важной особенностью является построчный вывод, где каждая запись сопровождается номером, что упрощает навигацию по списку. Например, при просмотре базы данных пользователь видит структурированный список студентов с полями: ФИО, факультет, средний балл, наличие льгот и размер стипендии. Это позволяет быстро оценить текущее состояние данных и проводить визуальный контроль информации.

Функция `addStudent` реализует добавление нового студента в базу данных. Процесс включает несколько этапов: чтение существующих данных, выделение места для добавления нового студента, запись в файл и очистка памяти. Сначала из файла считывается массив студентов с помощью `readStudentsFromFile`, чтобы сохранить уже имеющиеся записи. Для добавления места используется функция `ResizeArray`. Она увеличивает размер массива с определенным запасом – по степеням двойки. Такой подход снижает частоту перераспределения памяти при множественных добавлениях, повышая производительность. Обновленный массив

записывается обратно в файл через `writeStudentsToFile`, обеспечивая долгосрочное хранение данных. После завершения операции вызывается `deleteStudentsArray` для освобождения временно выделенной памяти.

Особое внимание уделено обработке пользовательского ввода. Например, при вводе таких данных, как средний балл или размер стипендии, проверяется, что введено именно число – для этого используется функция `isDigit`. Также контролируется форматирование строковых полей, таких как ФИО и факультет, чтобы исключить случайное введение некорректных символов или пустых значений.

Функция `changeStudentInfo` предоставляет возможность редактировать информацию о студенте в базе данных.

Алгоритм действий следующий:

1 Прежде чем начать редактирование, функция проверяет, существует ли студент с указанным номером. Если номер выходит за пределы допустимого диапазона, пользователю выводится понятное сообщение об ошибке – это помогает избежать некорректных обращений к массиву;

2 Глубокое копирование данных. Для замены старой записи на новую выполняется удаление объекта `Student` из массива и создание нового экземпляра с обновленными полями. При этом применяется конструктор копирования структуры `Student`, гарантирующий корректное управление динамическими строками;

3 Перезапись файла. Обновленный массив записывается обратно в файл, сохраняя изменения для последующих инструкций.

Функция учитывает возможные ошибки, такие как некорректный номер студента или поврежденный файл. Например, если файл не может быть открыт для чтения, функция завершает работу с соответствующим сообщением, предотвращая критические сбои или некорректное поведение программы.

Функция `deleteStudent` обеспечивает удаление студента из базы данных. Процесс включает: чтение исходного массива. Данные считываются из файла, чтобы сохранить существующие записи, создание нового массива. Формируется новый массив без элемента, соответствующего указанному номеру студента, перезапись файла. Обновленный массив записывается обратно, а временные данные очищаются с помощью `deleteStudentsArray`.

Для корректной работы с памятью реализованы два этапа очистки: удаление самого объекта `Student` и освобождение массива указателей. Это предотвращает утечки памяти даже при частых операциях добавления и удаления.

Функции `linearSearchByName` и `binarySearchByGrade` реализуют пользовательские запросы к базе данных. Линейный поиск работает с неотсортированными массивами, обеспечивая универсальность, а бинарный – требует предварительной сортировки через `SelectionSortByGrade`, но демонстрирует достаточную производительность. Результаты поиска выводятся в консоль с постраничным форматированием, где каждая запись сопровождается разделителями для улучшения читаемости.

Функции `quickSortByScholarship`, `selectionSortByGrade` и `insertionSortByName` реализуют пользовательские сортировки, интегрированные в меню программы.

Каждая из них:

– Считывает данные из файла;
– Выполняет сортировку с помощью соответствующего алгоритма;
– Перезаписывает отсортированный массив обратно в файл;
– Освобождает временную память.

Особенностью реализации является автоматическая перезапись файла после сортировки, что гарантирует сохранение изменений даже при внезапном завершении программы. Например, после сортировки студентов по стипендии (через `QuickSortByScholarship`) пользователь получает обновленный список, доступный для последующих операций поиска или фильтрации.

Функции `identifyIncreasedScholarship` и `scholarshipStatistics` предназначены для анализа данных и подготовки отчетов.

Первая функция (`identifyIncreasedScholarship`) сначала фильтрует студентов по критериям: минимальный балл и наличие льгот, после сортирует результаты по убыванию балла через `SelectionSortByGradeReverse` и в конце записывает отчет в текстовый файл result.txt.

Вторая функция (`scholarshipStatistics`) группирует студентов по факультетам (через `findFaculties`), рассчитывает минимальную и максимальную стипендию для каждой группы и формирует детализированный отчет с последующей сортировкой. Это позволяет быстро получить отчет с необходимыми параметрами для последующего анализа.

### 4.3 Функции поиска и анализа данных

Функции поиска и анализа данных необходимы для интерактивности и аналитических возможностей программы. Они позволяют пользователю эффективно находить информацию, фильтровать записи и формировать отчеты, что делает систему гибкой и более ориентированной на потребности конечного пользователя. Эти функции интегрированы в меню программы, обеспечивая доступ к операциям поиска и статистики через консольный интерфейс. Их реализация учитывает специфику работы с динамической памятью, необходимость преобразования данных и обработку исключений, связанных с вводом–выводом.

Функция `linearSearchByName`. Реализует линейный поиск студентов по точному совпадению ФИО. Используется для работы с неотсортированными данными, особенно при интерактивном поиске (например, в режиме реального времени). Алгоритм последовательно проверяет каждую запись, формируя список совпадений, которые выводятся в консоль с постраничным форматированием для улучшения читаемости.

Эта функция оптимальна для сценариев, где требуется найти конкретное совпадение (например, студента по имени), но не рекомендуется для массовых операций из–за низкой скорости на крупных массивах.

Функция `binarySearchByGrade` реализует бинарный поиск студентов по среднему баллу. В отличие от линейного метода, этот алгоритм требует предварительной сортировки данных по возрастанию балла с помощью `SelectionSortByGrade`. Такой функционал в лучшем случае позволяет использовать оптимальный метод поиска, временная сложность которого составляет O(log(n)), что значительно быстрее линейного аналога.

Алгоритм находит первый элемент, равный или превышающий целевой балл, а затем собирает все последующие записи с таким же значением. Однако ограничение на необходимость сортировки перед поиском делает функцию менее универсальной, чем линейный поиск. Тем не менее, в сценариях, где данные уже упорядочены или требуется многократный поиск, данный метод обеспечивает высокую производительность.

Функция `identifyIncreasedScholarship` предназначена для фильтрации студентов, соответствующих критериям повышения стипендии. Она учитывает минимальный средний балл и наличие льгот, что позволяет адаптировать выборку под нужды конечного пользователя. Например, преподаватель может запросить список студентов с баллом выше 8.0, включая только тех, кто имеет льготы, или всех без исключения. После фильтрации результаты сортируются по убыванию балла с помощью `SelectionSortByGradeReverse`, что позволяет выделить лидеров по успеваемости. Итоговый список записывается в текстовый файл result.txt, обеспечивая долгосрочное хранение данных для последующего анализа. Эта функция демонстрирует интеграцию нескольких модулей программы: фильтрации, сортировки и работы с файлами, что подчеркивает модульность архитектуры программы.

Функция `scholarshipStatistics` предназначена для анализа стипендий студентов по факультетам. Она группирует записи по полю `faculty`, рассчитывает минимальное и максимальное значения стипендий для каждой группы и формирует детализированный отчет. Например, пользователь может получить информацию о диапазоне стипендий на том и/или ином факультете, что может быть важно, к примеру, для планирования бюджета университета.

Реализация включает несколько этапов:

1 Группировка по факультетам. С помощью функции `findFaculties` определяются факультеты и исключается дублирование данных;

2 Фильтрация и анализ. Для каждого факультета вычисляются статистические показатели (минимум, максимум), а также формируется список студентов, соответствующих группе;

3 Запись отчета. Результаты сохраняются в текстовый файл result.txt, обеспечивая удобство последующего использования.

Эта функция демонстрирует комплексный подход к анализу данных, объединяя операции группировки, фильтрации и визуализации. Ее использование позволяет преподавателям и администраторам быстро оценивать финансовые показатели, выявлять аномалии и принимать обоснованные управленческие решения.

### 4.4 Управление памятью и файлами

Корректная работа с памятью и файлами крайне важна для стабильности программы [7]. В проекте реализованы функции, которые обеспечивают чтение и запись данных в бинарный файл Storage.bin, динамическое изменение массивов, например, при добавлении студентов, а также безопасное освобождение памяти после использования. Все эти операции интегрированы в модуль `FileManager`, что позволяет поддерживать целостность данных, минимизировать ошибки и оптимизировать производительность [8].

Функция `readStudentsFromFile` предназначена для восстановления данных о студентах из бинарного файла Storage.bin. Она обеспечивает точное воспроизведение структуры данных, сохраненной ранее, за счет последовательного считывания информации о каждом студенте.

Алгоритм работы включает следующие этапы:

4 Открытие файла. Проверка доступности файла для чтения. Если файл не найден или поврежден, функция возвращает ошибку и прекращает дальнейшие операции во избежание некорректной работы программы;

5 Чтение размера массива. Первой операцией считывается количество студентов, хранящееся в начале файла. Это значение определяет объем памяти, который необходимо выделить для временного хранения данных;

6 Восстановление массива студентов. Для каждого студента сначала считывается длина строковых полей (`fullName`, `faculty`), а затем сами строки. После этого извлекаются числовые значения (`averageGrade`, `hasPrivileges`, `scholarshipAmount`). Особое внимание уделено корректному копированию строк: для каждого поля выделяется память, соответствующая длине строки, и выполняется глубокое копирование данных.

7 Возврат данных. Восстановленный массив и его размер передаются в вызывающую функцию, после чего временные переменные освобождают память.

Функция `writeStudentsToFile` реализует запись массива студентов в бинарный файл Storage.bin.

Процесс записи включает:

1 Открытие файла. Файл открывается в режиме перезаписи, что позволяет оставлять только актуальные данные;

2 Запись размера массива. Перед самими данными записывается количество элементов массива, что позволяет функции чтения корректно определить объем памяти при последующем восстановлении;

3 Запись студентов. Для каждого студента последовательно записываются: длина строки `fullName` и само содержимое; длина строки `faculty` и её данные; числовые значения `averageGrade`, `hasPrivileges`, `scholarshipAmount`.

Функция активно используется при добавлении, удалении и редактировании записей, а также после сортировки или фильтрации данных.

Функция `ResizeArray` реализует механизм динамического изменения объема памяти, выделенной под массив студентов. Она применяется в случаях, когда необходимо расширить массив для добавления новых элементов или сократить его после удаления.

Основные особенности функции:

– Выделение памяти с запасом. Размер новой памяти вычисляется как степень двойки, превышающая текущий объем данных. Например, если текущий размер массива составляет 5 элементов, память выделяется под 8 (ближайшая степень двойки). Это снижает частоту операций перераспределения памяти при последовательных добавлениях, что повышает производительность;

– Копирование данных. Существующие элементы массива переносятся в новый массив, после чего старый массив удаляется [6]. При этом сохраняются указатели на объекты `Student`, а не их содержимое, что минимизирует вычислительные мощности на копирование;

– Гибкость использования. Функция применяется не только при добавлении студентов, но и при работе с результатами поиска, сортировки, получения статистики

Этот механизм значительно упрощает работу с динамическими структурами данных, такими как массивы, которые часто изменяются в процессе выполнения программы. Например, при добавлении нового студента функция `ResizeArray` автоматически увеличивает объем памяти, исключая необходимость ручного управления размером массива.

Функция `clearFile` предназначена для полной очистки содержимого файла Storage.bin. Она реализуется через открытие файла в режиме перезаписи, что приводит к удалению всех данных, хранящихся в нем. Особенностью функции является минимальный объем операций: после открытия файл сразу закрывается, что делает процесс быстрым и эффективным.

Данная функция находит применение в тестировании программы, где требуется сбросить базу данных до начального состояния, а также при выполнении операций массовой замены данных, когда старые записи больше не актуальны. Например, при добавлении данных из другого источника предварительная очистка файла гарантирует, что в хранилище не останется устаревших записей.

Все функции управления памятью и файлами реализуют механизмы обработки ошибок: проверка открытия файла, корректное освобождение памяти, контроль размера массива.

# 5 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

Разработанная программа представляет собой консольное приложение, предназначенное для управления базой данных студентов с возможностью сортировки, поиска, анализа статистики и редактирования записей. Интерфейс программы реализован через текстовое меню, обеспечивающее интуитивно понятную навигацию. В данном разделе подробно описаны сценарии использования программы, особенности взаимодействия с пользователем и обработка исключительных ситуаций, возникающих в ходе работы.

## 5.1 Интерфейс и основные функции

После запуска программы на экране отображается главное меню (рисунок 5.1.1), содержащий следующие опции:

– Выход из программы – завершение работы с сохранением всех изменений;
– База данных – операции с записями студентов;
– Поиск – поиск какой–либо информации;
– Сортировка – упорядочивание данных;
– Статистика – фильтрация студентов по критериям.

```
----- Instruction -----
0. Quit program
1. Database operations
2. Search
3. Sorting
4. Statistics
Choose an option:
```

Рисунок 5.1.1 – Внешний вид меню

Каждая опция имеет собственное подменю, которое активируется выбором соответствующего номера.

## 5.2 Работа с базой данных

При выборе пункта «Операции базы данных» пользователь получает доступ к следующим опциям (рисунок 5.2.1).

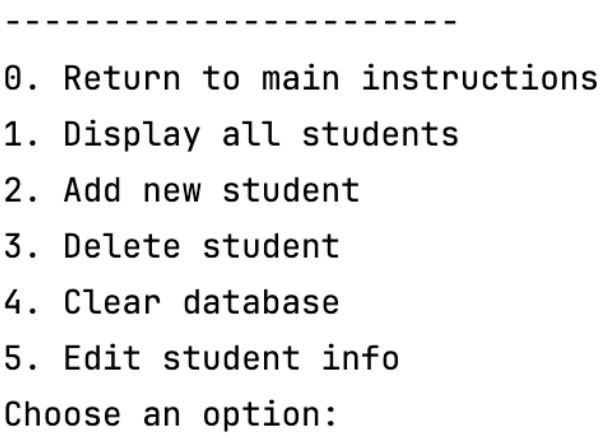


Рисунок 5.2.1 – Внешний вид меню «Операции базы данных»

Каждой опции соответствует функция, предназначенная для прямой работы с базой данных.

Например, опция «Отображение всех студентов» запускает функцию, считывает данные из файла Storage.bin, а после выводит список студентов в консоль. Каждая запись включает: ФИО, факультет, средний балл, наличие льгот, размер стипендии. Если файл пуст или поврежден, программа выводит сообщение об ошибке: «Файл пуст: Storage.bin» или «Ошибка открытия файла».

Опция «Добавить нового студента» запускает функцию, требующую от пользователя следующего ввода данных:

- ФИО (например, «Иванов Иван Иванович»);
- Факультет (например, «ФИТУ»);
- Средний балл (положительное либо нулевое число);
- Наличие льгот (1 – да, 0 – нет);
- Размер стипендии (положительное либо нулевое число).

Программа также проверит данные и покажет ошибки при некорректном вводе. Если данные введены правильно – студент успешно добавиться в базу данных.

Опция «Удалить студента» запускает функцию, требующую указать номер студента из списка. Если файл поврежден, номер введен неверно или же номер студента выходит за список, выводится сообщение: «Не удалось удалить студента». Если все данные введены верно – студент будет удален из базы данных. Функция не требует подтверждения, поэтому рекомендуется использовать опцию «Отобразить всех студентов» перед удалением и проверить правильность введенных данных.

Опция «Очистить базу данных» вызывает функцию, которая перезаписывает файл Storage.bin, удаляя все записи. Перед выполнением действия программа предлагает подтверждение: «Подтверждение (1 – да, 0 – нет)», что позволяет исключить риск случайного удаления базы данных.

Опция «Редактировать информацию об студенте» вызывает функцию, которая позволяет изменить любое поля информации о студенте. Программа требует ввести номер студента, после чего предварительно отображает текущие данные и предлагает ввести новые значения. Если пользователь нажимает Enter, соответствующее поле остается без изменений. После ввода всех данных программа запрашивает подтверждение изменений: «Подтверждение изменений (1 – да, 0 – нет)». Если пользователь подтвердит изменения и программа не обнаружит ошибок в вводе – информация у студента измениться. Это позволяет избежать ошибок, связанных с некорректным вводом.

### 5.3 Поиск данных

При выборе пункта «Операции базы данных» пользователь получает доступ к следующим опциям (рисунок 5.3.1).

```
----------------------
0. Return to main instructions
1. Linear search by name
2. Binary search by grade
Choose an option:
```

Рисунок 5.3.1 – Внешний вид меню «Поиск»

Каждой опции соответствует функция, предназначенная для поиска информации о студенте в базе данных.

Так опция «Линейный поиск по имени» вызывает функцию, которая заходит в файл Storage.bin и начинает поиск по ФИО, предварительно введенным пользователем. Если поиск удался, пользователь увидит информацию о студентах, иначе – ошибку «Студент не найден».

Опция «Бинарный поиск по среднему баллу» вызывает функцию, которая заходит в файл Storage.bin и начинает поиск по среднему баллу, предварительно введенным пользователем. Если поиск удался, пользователь увидит информацию о студентах, иначе – ошибку «Студент не найден».

### 5.4 Сортировка

При выборе пункта «Сортировка» пользователь получает доступ к следующим опциям (рисунок 5.4.1).

```
----------------------
0. Return to main instructions
1. Quick sort by scholarship
2. Selection sort by grade
3. Insertion sort by name
Choose an option:
```

Рисунок 5.4.1 – Внешний вид меню «Сортировка»

Программа предоставляет три алгоритма сортировки: быстрая сортировка по размеру стипендии, сортировка выбором по среднему баллу и сортировка вставками по ФИО.

После каждой сортировки данные автоматически сохраняются в файл Storage.bin, что может повлиять на скорость последующих операций (например, бинарному поиску). Если файл пуст, сортировка невозможна, и программа выводит: «Файл пуст: Storage.bin».

### 5.5 Статистический анализ

При выборе пункта «Статистика» пользователь получает доступ к следующим опциям (рисунок 5.5.1).

```
----------------------
0. Return to main instructions
1. Identify students for an increased scholarship
2. Scholarship statistics
Choose an option:
```

Рисунок 5.5.1 – Внешний вид меню «Статистика»

Каждой опции соответствует функция, предназначенная для получения статистики о студентах в базе данных.

Опция «Определение студентов для повышенной стипендии» вызывает функцию, которая позволяет выбрать студентов, соответствующих двум критериям: средний балл выше значения пользователя и указания наличия льгот. Результаты сортируются по убыванию балла и сохраняются в файл result.txt. Если подходящих записей нет, программа выводит: «Студенты не найдены».

Опция «Статистика стипендий» вызывает функцию, которая рассчитывает минимальную и максимальную стипендию для каждого факультета, группируя данные из Storage.bin.

Результаты записываются в result.txt в формате: наименование факультета, минимальная и максимальная стипендии на факультете, вывод

студентов. Если файл Storage.bin пуст, выводится сообщение: «Файл пуст: Storage.bin».

### 5.6 Пользовательский опыт

Интерфейс программы спроектирован с учетом принципов минимальной нагрузки на пользователя:

– Подсказки в консоли – каждая операция сопровождается запросами с примерами ввода;

– Цветовая индикация – поля данных окрашены в зеленый цвет, а номера студентов по счету в синий цвет, что упрощает визуальный анализ;

– Постраничный вывод – длинные списки студентов отображаются с разделителями «————————————», улучшающими восприятие.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана функциональная программа на языке C++, предназначенная для управления базой данных студентов. Программа предоставляет широкий спектр возможностей: добавление, удаление и редактирование записей, сортировка данных по различным критериям, поиск информации, а также анализ статистики стипендий и формирование отчетов.

В рамках проекта были изучены и применены различные алгоритмы сортировки и поиска, адаптированные под специфику обработки данных в виде массивов указателей на объекты типа `Student`. Были реализованы и протестированы такие алгоритмы сортировки, как быстрая сортировка, сортировка выбором и вставками. Для поиска данных использовались линейный и бинарный методы.

Программа включает интерфейс командной строки, который обеспечивает удобное взаимодействие с пользователем. Все данные о студентах сохраняются в бинарном файле Storage.bin, а результаты работы программы сохраняются в result.txt, что делает возможным их использование во внешних системах. Также предусмотрена корректная работа с динамической памятью, исключающая утечки и ошибки при манипуляции с большими объемами данных.

Большое внимание было уделено обработке исключительных ситуаций, таких как некорректный ввод пользователя, отсутствие необходимых данных, ошибки чтения/записи файлов. В программе реализованы механизмы проверки ввода числовых значений, подтверждения действий пользователя.

Результатом проекта стало консольное приложение, которое может быть использовано в образовательных учреждениях для автоматизации операций с данными студентов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Habr [Электронный ресурс] – Описание алгоритмов сортировки и сравнение их... / Хабр. – Режим доступа: https://habr.com/ru/articles/335920/

[2] CyberForum [Электронный ресурс] – Форум программистов C++. – Электронные данные. – Режим доступа: http://www.cyberforum.ru/cpp/

[3] CPP–REFERENCE [Электронный ресурс]. – Режим доступа: https://en.cppreference.com/w/

[4] Metanit [Электронный ресурс] – Руководство по программированию на языке C++. – Режим доступа: https://metanit.com/cpp/tutorial/?ysclid=mazbti6lx533922633

[5] Habr [Электронный ресурс] – Алгоритмы поиска в строке / Хабр. – Режим доступа: https://habr.com/ru/articles/111449/

[6] Ravesli [Электронный ресурс] – Динамические массивы в C++ / Ravesli. – Режим доступа: https://ravesli.com/urok–86–dinamicheskie–massivy/?ysclid=mazc0celfx485898062

[7] Github [Электронный ресурс] – Read and write binary files in C++ | Simon Lizotte. – Режим доступа: https://siliz4.github.io/guides/tutorials/2020/05/21/guide–on–binary–files–cpp.html

[8] Github [Электронный ресурс] – C++ | Namespaces. – Режим доступа: https://devtut.github.io/cpp/namespaces.html

[9] Geeksforgeeks [Электронный ресурс] – Structures in C++ | GeeksforGeeks. – Режим доступа: https://www.geeksforgeeks.org/structures–in–cpp/?ysclid=mazc54idsl593426706

# ПРИЛОЖЕНИЕ А
# (ОБЯЗАТЕЛЬНОЕ)
## Листинг кода

```
FileManager.h
#ifndef FILEMANAGER_H
#define FILEMANAGER_H

#include "Student.h"
#include <iostream>
namespace FileManager {

    void ResizeArray(Student **&students, size_t size);
    // Read students and write it to array
    bool readStudentsFromFile(const char* filename,
Student**& students, size_t& size);
    // Write students to file
    bool writeStudentsToFile(const char* filename, Student**
students, size_t size);
    // Clear file from path
    void clearFile(const char* filename);

    bool writeIncreasedScholarshipStatisticsToFile(const
char* filename, Student** students, size_t size);
    bool writeScholarshipStatisticsToFile(const char*
filename, Student** students,
        size_t size, char* faculty, double minS, double
maxS);

    size_t getNumberOfStudentsInFile(const char* filename);
}


#endif // FILEMANAGER_H


Operations.h
#ifndef OPERATIONS_H
#define OPERATIONS_H

#include "Student.h"
#include <iostream>

namespace Search {
    void LinearSearchByName(Student** inputStudents, size_t
inputSize,
        Student**& outputStudents, size_t& outputSize, const
char* name);

    void BinarySearchByGrade(Student** inputStudents, size_t
inputSize,
```

```
    Student**& outputStudents, size_t& outputSize, double
grade);

    namespace Sorting {
        void QuickSortByScholarship(Student**& students, size_t
low, size_t high);
        size_t QuickSortPartition(Student**& students, size_t
low, size_t high);

        void QuickSortByScholarshipReverse(Student**& students,
size_t low, size_t high);
        size_t QuickSortPartitionReverse(Student**& students,
size_t low, size_t high);

        void SelectionSortByGrade(Student**& students, size_t
size);
        void SelectionSortByGradeReverse(Student**& students,
size_t size);

        void InsertionSortByName(Student**& students, size_t
size);
    }

    namespace AdditionalFunctions {
        void findFaculties(Student** inputStudents, size_t
inputSize, char**& faculties, size_t& facultiesSize);
        bool isDigit(char* c);
        void deleteStudentsArray(Student** inputStudents, size_t
inputSize);
        bool compareChars(const char* a, const char* b);
    }

    #endif // OPERATIONS_H

    Database.h
    #ifndef DATABASE_H
    #define DATABASE_H

    #include <iostream>
    #include <fstream>
    #include "Student.h"

    namespace Database {

        void displayAll(const char* filename);
        bool addStudent(const char* filename, Student* student);
        bool changeStudentInfo(const char* filename, Student*
student, int studentNumber);
        bool deleteStudent(const char* filename, int
studentNumber);
        size_t getNumOfStudents(const char* filename);
```

## Продолжение приложения А

```
    void linearSearchByName(const char* filename, const char*
name);
        void binarySearchByGrade(const char* filename, double
grade);

        void quickSortByScholarship(const char* filename);
        void selectionSortByGrade(const char* filename);
        void insertionSortByName(const char* filename);

        void identifyIncreasedScholarship(const char*
storageFilename, const char* resultFilename,
            double minGrade, bool needPrivileges);

        void scholarshipStatistics(const char* storageFilename,
const char* resultFilename);
    }

    #endif


    Student.h
    #ifndef STUDENT_H
    #define STUDENT_H
    #include <fstream>

    struct Student {
    private:
        char *fullName;
        char *faculty;
        double averageGrade;
        bool hasPrivileges;
        double scholarshipAmount;

    public:
        // Constructor
        explicit Student(char *fullName, char *faculty,
            double averageGrade, bool hasPrivileges, double
scholarshipAmount);
        // Destructor
        ~Student();

        // Setters
        void setFullName(char *fullName);
        void setFaculty(char *faculty);
        void setAverageGrade(double grade);
        void setHasPrivileges(bool hasPrivileges);
        void setScholarshipAmount(double scholarshipAmount);

        // Getters
        char* getFullName();
        char* getFaculty();
```

## Продолжение приложения А

```
    double getAverageGrade();
        bool getHasPrivileges();
        double getScholarshipAmount();

        // Other func
        void print();
        void print(std::ofstream &file);
        Student(const Student& other); // Copying constructor
        Student& operator=(const Student& other); // Assignment
operator
    };

    #endif //STUDENT_H


    Student.cpp
    #include "Student.h"
    #include <cstring>
    #include <iostream>
    #include <ostream>

    #define GREEN_COLOR "\033[32m"
    #define RESET_COLOR "\x1B[0m"

    // Constructor
    Student::Student(char *fullName, char *faculty, double
averageGrade, bool hasPrivileges, double scholarshipAmount) {
        // Allocate memory for fullName and copy the content

        this->faculty = nullptr;
        if (fullName != nullptr) {
            this->fullName = new char[strlen(fullName) + 1];
            strcpy(this->fullName, fullName);
        }

        // Allocate memory for faculty and copy the content
        if (faculty != nullptr) {
            this->faculty = new char[strlen(faculty) + 1];
            strcpy(this->faculty, faculty);
        }

        // Initialize other fields
        this->averageGrade = averageGrade;
        this->hasPrivileges = hasPrivileges;
        this->scholarshipAmount = scholarshipAmount;
    }

    // Destructor
    Student::~Student() {
        // Free dynamically allocated memory
        delete[] fullName;
```

## Продолжение приложения А

```
delete[] faculty;
}

#pragma region Setters

void Student::setFullName(char *fullName) {
    // Free existing memory before allocating new memory
    delete[] this->fullName;
    this->fullName = new char[strlen(fullName) + 1];
    strcpy(this->fullName, fullName)
}

void Student::setFaculty(char *faculty) {
    // Free existing memory before allocating new memory
    delete[] this->faculty;
    this->faculty = new char[strlen(faculty) + 1];
    strcpy(this->faculty, faculty);
}

void Student::setAverageGrade(double grade) {
    this->averageGrade = grade;
}

void Student::setHasPrivileges(bool hasPrivileges) {
    this->hasPrivileges = hasPrivileges;
}

void Student::setScholarshipAmount(double scholarshipAmount)
{
    this->scholarshipAmount = scholarshipAmount;
}

#pragma endregion

#pragma region Getters

char* Student::getFullName() {
    return fullName;
}

char* Student::getFaculty() {
    return faculty;
}

double Student::getAverageGrade() {
    return averageGrade;
}

bool Student::getHasPrivileges() {
    return hasPrivileges;
}
```

## Продолжение приложения А

```
    double Student::getScholarshipAmount() {
        return scholarshipAmount;
    }

    #pragma endregion

    void Student::print() {
        std::cout << RESET_COLOR << "Full Name: " << GREEN_COLOR
<< this->fullName << std::endl;
        std::cout << RESET_COLOR << "Faculty: " << GREEN_COLOR
<< this->faculty << std::endl;
        std::cout << RESET_COLOR << "Average Grade: " <<
GREEN_COLOR << this->averageGrade << std::endl;
        std::cout << RESET_COLOR << "Has Privileges: " <<
GREEN_COLOR << this->hasPrivileges << std::endl;
        std::cout << RESET_COLOR << "Scholar Amount: " <<
GREEN_COLOR << this->scholarshipAmount << std::endl;
        std::cout << RESET_COLOR;
    }

    void Student::print(std::ofstream &file) {
        file << "Full Name: " << this->fullName << std::endl;
        file << "Faculty: " << this->faculty << std::endl;
        file << "Average Grade: " << this->averageGrade <<
std::endl;
        file << "Has Privileges: " << this->hasPrivileges <<
std::endl;
        file << "Scholar Amount: " << this->scholarshipAmount <<
std::endl;
    }

    Student::Student(const Student& other) {
        fullName = new char[strlen(other.fullName) + 1];
        strcpy(fullName, other.fullName);
        faculty = new char[strlen(other.faculty) + 1];
        strcpy(faculty, other.faculty);
        averageGrade = other.averageGrade;
        hasPrivileges = other.hasPrivileges;
        scholarshipAmount = other.scholarshipAmount;
    }

    Student& Student::operator=(const Student& other) {
        if (this != &other) {
            delete[] fullName;
            delete[] faculty;
            fullName = new char[strlen(other.fullName) + 1];
            strcpy(fullName, other.fullName);
            faculty = new char[strlen(other.faculty) + 1];
            strcpy(faculty, other.faculty);
            averageGrade = other.averageGrade;
            hasPrivileges = other.hasPrivileges;
```

```
    scholarshipAmount = other.scholarshipAmount;
        }
        return *this;
    }

    FileManager.cpp
    #include "FileManager.h"
    #include <fstream>
    #include <cstring>
    #include <iostream>

    void FileManager::ResizeArray(Student**& students, size_t
size) {
        if (size == 0) return;
        size_t capacity = 1;
        while (capacity <= size) capacity <<= 1;

        Student** newStudents = new Student*[capacity];
        if (students)
            for (size_t i = 0; i < size; i++) {
                newStudents[i] = students[i];
            }

        // if (students)
        //     delete[] students; // Delete the old array, not
individual elements yet

        students = newStudents;
    }

    bool FileManager::readStudentsFromFile(const char *filename,
Student **&students, size_t &size) {
        std::ifstream file (filename, std::ifstream::in);
        if (!file.is_open()) {
            std::cerr << "Could not open file " << filename <<
std::endl;
            return false;
        }

        // Read array size
        file.read(reinterpret_cast<char*>(&size),
sizeof(size_t));

        if (size == 0)
            return students = nullptr, false;
        ResizeArray(students, size);
        //students = new Student*[size];

        for (size_t i = 0; i < size; i++) {
            // Read size of fName
            size_t fullNameLength;
```

## Продолжение приложения А

```
            file.read(reinterpret_cast<char*>(&fullNameLength),
sizeof(fullNameLength));
            char* fullName = new char[fullNameLength];
            file.read(fullName, fullNameLength);

            // Read size if faculty
            size_t facultyLength;
            file.read(reinterpret_cast<char*>(&facultyLength),
sizeof(facultyLength));
            char* faculty = new char[facultyLength];
            file.read(faculty, facultyLength);

            // Read average grade
            double averageGrade;
            file.read(reinterpret_cast<char*>(&averageGrade),
sizeof(averageGrade));

            // Read privileges
            bool hasPrivileges;
            file.read(reinterpret_cast<char*>(&hasPrivileges),
sizeof(hasPrivileges));

            // Read scholarship amount
            double scholarshipAmount;

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

            // Add info to array
            students[i] = new Student(fullName, faculty,
averageGrade, hasPrivileges, scholarshipAmount);
        }

        return true;
    }

    bool FileManager::writeStudentsToFile(const char *filename,
Student **students, size_t size) {
        std::ofstream file (filename, std::ofstream::out);
        if (!file.is_open()) {
            std::cerr << "Could not open file " << filename <<
std::endl;
            return false;
        }

        file.write(reinterpret_cast<char*>(&size),
sizeof(size_t));
        for (size_t i = 0; i < size; i++) {
            size_t fullNameLength = strlen(students[i]-
>getFullName());
```

## Продолжение приложения А

```
     file.write(reinterpret_cast<char*>(&fullNameLength),
sizeof(fullNameLength));
             file.write(students[i]->getFullName(),
fullNameLength);

             size_t facultyLength = strlen(students[i]-
>getFaculty());
             file.write(reinterpret_cast<char*>(&facultyLength),
sizeof(facultyLength));
             file.write(students[i]->getFaculty(),
facultyLength);

             double averageGrade = students[i]-
>getAverageGrade();
             file.write(reinterpret_cast<char*>(&averageGrade),
sizeof(averageGrade));

             bool hasPrivileges = students[i]-
>getHasPrivileges();
             file.write(reinterpret_cast<char*>(&hasPrivileges),
sizeof(hasPrivileges));

             double scholarshipAmount = students[i]-
>getScholarshipAmount();

file.write(reinterpret_cast<char*>(&scholarshipAmount),
sizeof(scholarshipAmount));
         }

         return true;
     }

     void FileManager::clearFile(const char *filename) {
         std::ofstream file(filename);
         file.close();
     }

     bool
FileManager::writeIncreasedScholarshipStatisticsToFile(const
char *filename, Student **students, size_t size) {
         std::ofstream file(filename);

         if (!file.is_open()) {
             std::cerr << "Could not open file " << filename <<
std::endl;
             return false;
         }

         file << size << " students:" << std::endl;
         for (int i = 0; i < size; i++) {
             file << "----------------------" << std::endl;
```

```
    students[i]->print(file);
        }

        file.close();
        return true;
    }

    bool FileManager::writeScholarshipStatisticsToFile(const
char *filename, Student **students,
        size_t size, char *faculty, double minS, double maxS) {

        std::ofstream file(filename, std::ofstream::app);
        if (!file.is_open()) {
            std::cerr << "Could not open file " << filename <<
std::endl;
            return false;
        }

        file << std::endl;
        file << "----- " << faculty << " -----" << std::endl;
        file << "Min scholarship: " << minS << std::endl;
        file << "Max scholarship: " << maxS << std::endl;
        for (int i = 0; i < size; i++) {
            file << "----------------------" << std::endl;
            students[i]->print(file);
        }

        file.close();
        return true;
    }

    size_t FileManager::getNumberOfStudentsInFile(const char
*filename) {
        std::ifstream file(filename, std::ios::binary);
        if (!file.is_open()) {
            std::cerr << "Could not open file " << filename <<
std::endl;
            return 0;
        }
        size_t numberOfStudents;
        file.read(reinterpret_cast<char*>(&numberOfStudents),
sizeof(numberOfStudents));
        return numberOfStudents;
    }


    Operations.cpp
    #include "Operations.h"

    #pragma region Search
```

## Продолжение приложения А

```
    void Search::LinearSearchByName(Student **inputStudents,
size_t inputSize,
        Student **&outputStudents, size_t &outputSize, const
char *name) {

        outputStudents = new Student*[inputSize];
        outputSize = 0;

        for (size_t i = 0; i < inputSize; i++) {
            if (strcmp(inputStudents[i]->getFullName(), name) ==
0) {
                outputStudents[outputSize++] = inputStudents[i];
            }
        }
    }

    void Search::BinarySearchByGrade(Student **inputStudents,
size_t inputSize,
        Student **&outputStudents, size_t &outputSize, double
grade) {

        size_t left = 0, right = inputSize - 1;
        while (left < right) {

            size_t mid = (left + right) / 2;

            if (inputStudents[mid]->getAverageGrade() < grade) {
                left = mid + 1;
            }
            else{
                right = mid;
            }
        }

        outputSize = 0;
        for (size_t i = left; i < inputSize; i++) {
            if (inputStudents[i]->getAverageGrade() == grade) {
                outputStudents[outputSize++] = inputStudents[i];
            }
            else break;
        }
    }

    #pragma endregion

    #pragma region Sorting

    size_t Sorting::QuickSortPartition(Student **&arr, size_t
low, size_t high) {

        double pivot = arr[high]->getScholarshipAmount();
```

## Продолжение приложения А

```
    size_t i = low - 1;

        for (size_t j = low; j <= high - 1; j++) {
            if (arr[j]->getScholarshipAmount() < pivot)
                std::swap(arr[++i], arr[j]);
        }

        std::swap(arr[i + 1], arr[high]);
        return i + 1;
    }
    void Sorting::QuickSortByScholarship(Student **&arr, size_t
low, size_t high) {
        if (low < high) {

            size_t pi = QuickSortPartition(arr, low, high);

            QuickSortByScholarship(arr, low, pi - 1);
            QuickSortByScholarship(arr, pi + 1, high);
        }
    }

    size_t Sorting::QuickSortPartitionReverse(Student **&arr,
size_t low, size_t high) {
        double pivot = arr[high]->getScholarshipAmount();

        size_t i = low - 1;

        for (size_t j = low; j <= high - 1; j++) {
            if (arr[j]->getScholarshipAmount() > pivot)
                std::swap(arr[++i], arr[j]);
        }

        std::swap(arr[i + 1], arr[high]);
        return i + 1;
    }
    void Sorting::QuickSortByScholarshipReverse(Student **&arr,
size_t low, size_t high) {
        if (low < high) {

            size_t pi = QuickSortPartitionReverse(arr, low,
high);

            QuickSortByScholarshipReverse(arr, low, pi - 1);
            QuickSortByScholarshipReverse(arr, pi + 1, high);
        }
    }

    void Sorting::SelectionSortByGrade(Student **&arr, size_t
size) {

        for (size_t i = 0; i < size - 1; i++) {
```

## Продолжение приложения А

```
            size_t minIndex = i;
             for (size_t j = i + 1; j < size; j++) {
                 if (arr[j]->getAverageGrade() < arr[minIndex]-
>getAverageGrade()) {
                     minIndex = j;
                 }
             }
             std::swap(arr[i], arr[minIndex]);
         }

     }

     void Sorting::SelectionSortByGradeReverse(Student **&arr,
size_t size) {
         for (size_t i = 0; i < size - 1; i++) {
             size_t minIndex = i;
             for (size_t j = i + 1; j < size; j++) {
                 if (arr[j]->getAverageGrade() > arr[minIndex]-
>getAverageGrade()) {
                     minIndex = j;
                 }
             }
             std::swap(arr[i], arr[minIndex]);
         }
     }

     void Sorting::InsertionSortByName(Student **&arr, size_t
size) {
         for (size_t i = 1; i < size; ++i) {
             Student* key = arr[i];
             long long j = i - 1;

             // Сравнение строк через `std::string`
             while (j >= 0 &&
AdditionalFunctions::compareChars(arr[j]->getFullName(), key-
>getFullName())) {
                 arr[j + 1] = arr[j];
                 --j;
             }

             arr[j + 1] = key;
         }
     }


     #pragma endregion

     #pragma region Additional functions

     void AdditionalFunctions::findFaculties(Student **arr,
size_t size, char **&faculties,
```

## Продолжение приложения А

```
    size_t &facultiesSize) {
        faculties = new char*[size];
        facultiesSize = 0;

        faculties[facultiesSize++] = arr[0]->getFaculty();
        for (size_t i = 1; i < size; i++) {
            bool found = false;
            char* faculty = arr[i]->getFaculty();
            for (size_t j = 0; j < facultiesSize; j++) {
                if (strcmp(arr[i]->getFaculty(), faculties[j])
== 0) {
                    found = true;
                    break;
                }
            }
            if (!found) {
                faculties[facultiesSize++] = arr[i]-
>getFaculty();
            }
        }
    }

    bool AdditionalFunctions::isDigit(char *c) {
        size_t size = strlen(c);
        for (size_t i = 0; i < size; i++) {
            if (!((c[i] >= '0' && c[i] <= '9') || (c[i] == ',')
|| (c[i] == '.')))
                return false;
        }
        return true;
    }

    void AdditionalFunctions::deleteStudentsArray(Student
**inputStudents, size_t inputSize) {
        if (inputStudents == nullptr || inputSize == 0)
            return;

        for (size_t i = 0; i < inputSize; i++) {
            delete inputStudents[i];
        }

        delete[] inputStudents;
    }

    bool AdditionalFunctions::compareChars(const char* a, const
char* b) {
        if (a == b) return 0;       // Same pointer or both null
        if (!a) return -1;          // `a` is null (treated as
smaller)
        if (!b) return 1;           // `b` is null (treated as
smaller)
```

```
       while (*a && *b && *a == *b) {
              a++;
              b++;
          }
          return (*a - *b) > 0;
      }

      #pragma endregion


      Database.cpp
      #include "Database.h"
      #include "FileManager.h"
      #include "Operations.h"
      #define BLUE_COLOR "\033[34m"
      #define RESET_COLOR "\033[0m"

      void Database::displayAll(const char *filename) {
          size_t size = 0;
          Student **students = nullptr;

          if (!FileManager::readStudentsFromFile(filename,
students, size)) {
              std::cerr << "File is empty: " << filename <<
std::endl;
              return;
          }

          for (size_t i = 0; i < size; i++) {
              std::cout << BLUE_COLOR << i + 1 << RESET_COLOR <<
". ";
              students[i]->print();
              delete students[i];
          }

          delete[] students;
      }

      bool Database::addStudent(const char *filename, Student
*student) {
          Student **students = nullptr;
          size_t size = 0;

          FileManager::readStudentsFromFile(filename, students,
size);

          FileManager::ResizeArray(students, size + 1);
          students[size] = student;

          FileManager::writeStudentsToFile(filename, students,
size + 1);
```

```
        AdditionalFunctions::deleteStudentsArray(students, size +
1);

        return true;
    }

    bool Database::changeStudentInfo(const char* filename,
Student* student, int studentNumber) {
        Student** students = nullptr;
        size_t size = 0;

        if (!FileManager::readStudentsFromFile(filename,
students, size)) {
            std::cerr << "File error: " << filename <<
std::endl;
            return false;
        }

        if (size == 0) {
            std::cerr << "File is empty: " << filename <<
std::endl;
            return false;
        }

        studentNumber--; // Convert to zero-based index

        if (studentNumber >= size) {
            std::cerr << "Student number out of range.\n";
            AdditionalFunctions::deleteStudentsArray(students,
size);
            return false;
        }

        // Safely delete the old student
        delete students[studentNumber];
        // Replace with the updated student
        students[studentNumber] = student;

        // Write the updated array back to file
        bool success =
FileManager::writeStudentsToFile(filename, students, size);
        AdditionalFunctions::deleteStudentsArray(students,
size);

        return success;
    }

    bool Database::deleteStudent(const char* filename, int
studentNumber) {
        Student** students = nullptr;
        size_t size = 0;
```

## Продолжение приложения А

```
     // Step 1: Read all students from the file
          if (!FileManager::readStudentsFromFile(filename,
students, size)) {
              std::cerr << "Error opening file: " << filename <<
"\n";
              return false;
          }

          if (size == 0) {
              std::cerr << "File is empty: " << filename << "\n";
              AdditionalFunctions::deleteStudentsArray(students,
size);
              return false;
          }

          // Convert to 0-based index
          studentNumber--;

          // Step 2: Validate student number
          if (studentNumber < 0 ||
static_cast<size_t>(studentNumber) >= size) {
              std::cerr << "Invalid student number.\n";
              AdditionalFunctions::deleteStudentsArray(students,
size);
              return false;
          }

          // Step 3: Extract the student to delete
          Student* toDelete = students[studentNumber];

          // Step 4: Create a new array excluding the deleted
student
          Student** newStudents = new Student*[size - 1];
          size_t j = 0;
          for (size_t i = 0; i < size; ++i) {
              if (i != static_cast<size_t>(studentNumber)) {
                  newStudents[j++] = students[i];
              }
          }

          // Step 5: Rewrite the file with the updated array
          bool success =
FileManager::writeStudentsToFile(filename, newStudents, size -
1);

          // Step 6: Clean up memory
          delete toDelete;
          AdditionalFunctions::deleteStudentsArray(newStudents,
size - 1);
          AdditionalFunctions::deleteStudentsArray(students,
size);
```

```
        return success;
    }

    size_t Database::getNumOfStudents(const char *filename) {
        size_t size =
FileManager::getNumberOfStudentsInFile(filename);
        if (size == 0) {
            std::cerr << "Find 0 students. Database is empty" <<
std::endl;
            return 0;
        }
        return size;
    }

    void Database::linearSearchByName(const char *filename,
const char *name) {
        Student **students = nullptr;
        size_t size = 0;

        if (!FileManager::readStudentsFromFile(filename,
students, size)) {
            std::cerr << "Find 0 students. Database is empty" <<
std::endl;
            return;
        }

        Student **outputStudents = nullptr;
        size_t outputSize = 0;

        Search::LinearSearchByName(students, size,
outputStudents, outputSize, name);

        if (outputSize == 0) {
            std::cerr << "No students found" << std::endl;
            return;
        }

        std::cout << outputSize << " students found:" <<
std::endl;
        for (size_t i = 0; i < outputSize; i++) {
            std::cout << "----------------------" << std::endl;
            outputStudents[i]->print();
        }

        AdditionalFunctions::deleteStudentsArray(students,
size);
    }

    void Database::binarySearchByGrade(const char *filename,
double grade) {
        Student **students = nullptr;
```

## Продолжение приложения А

```
      size_t size = 0;

          if (!FileManager::readStudentsFromFile(filename,
students, size)) {
              std::cerr << "Find 0 students. Database is empty" <<
std::endl;
              return;
          }
          Sorting::SelectionSortByGrade(students, size);

          Student **outputStudents = new Student*[size];
          size_t outputSize = 0;

          Search::BinarySearchByGrade(students, size,
outputStudents, outputSize, grade);

          if (outputSize == 0) {
              std::cerr << "No students found" << std::endl;
              return;
          }

          std::cout << outputSize << " students found:" <<
std::endl;
          for (size_t i = 0; i < outputSize; i++) {
              std::cout << "----------------------" << std::endl;
              outputStudents[i]->print();
          }

          AdditionalFunctions::deleteStudentsArray(students,
size);
      }

     void Database::quickSortByScholarship(const char *filename)
{
          Student **students = nullptr;
          size_t size = 0;

          if (!FileManager::readStudentsFromFile(filename,
students, size)) {
              std::cerr << "File is empty: " << filename <<
std::endl;
              return;
          }

          Sorting::QuickSortByScholarship(students, 0, size - 1);

          FileManager::writeStudentsToFile(filename, students,
size);

          AdditionalFunctions::deleteStudentsArray(students,
size);
```

```
    }

    void Database::selectionSortByGrade(const char *filename) {
        Student **students = nullptr;
        size_t size = 0;

        if (!FileManager::readStudentsFromFile(filename,
students, size)) {
            std::cerr << "File is empty: " << filename <<
std::endl;
            return;
        }

        Sorting::SelectionSortByGrade(students, size);

        FileManager::writeStudentsToFile(filename, students,
size);

        AdditionalFunctions::deleteStudentsArray(students,
size);
    }

    void Database::insertionSortByName(const char *filename) {
        Student **students = nullptr;
        size_t size = 0;

        if (!FileManager::readStudentsFromFile(filename,
students, size)) {
            std::cerr << "File is empty: " << filename <<
std::endl;
            return;
        }

        Sorting::InsertionSortByName(students, size);

        FileManager::writeStudentsToFile(filename, students,
size);

        AdditionalFunctions::deleteStudentsArray(students,
size);
    }

    void Database::identifyIncreasedScholarship(const char
*storageFilename, const char *resultFilename,
        double minGrade, bool needPrivileges) {
        Student **students = nullptr;
        size_t size = 0;

        if (!FileManager::readStudentsFromFile(storageFilename,
students, size)) {
```

```
     std::cerr << "File is empty: " << storageFilename <<
std::endl;
             return;
         }

         Student **outputStudents = new Student*[size];
         size_t outputSize = 0;
         for (size_t i = 0; i < size; i++) {
             if (students[i]->getAverageGrade() >= minGrade)
                 if (!needPrivileges || (needPrivileges &&
needPrivileges == students[i]->getHasPrivileges()))
                     outputStudents[outputSize++] = students[i];
         }

         if (outputSize == 0) {
             std::cerr << "No students found" << std::endl;
             return;
         }

         Sorting::SelectionSortByGradeReverse(outputStudents,
outputSize);


FileManager::writeIncreasedScholarshipStatisticsToFile(resultFil
ename, outputStudents, outputSize);

         AdditionalFunctions::deleteStudentsArray(students,
size);
     }

     void Database::scholarshipStatistics(const char
*storageFilename, const char *resultFilename) {
         Student **students = nullptr;
         size_t size = 0;

         if (!FileManager::readStudentsFromFile(storageFilename,
students, size)) {
             std::cerr << "File is empty: " << storageFilename <<
std::endl;
             return;
         }
         FileManager::clearFile(resultFilename);

         char** faculties;
         size_t facultiesSize = 0;
         AdditionalFunctions::findFaculties(students, size,
faculties, facultiesSize);

         for (size_t i = 0; i < facultiesSize; i++) {
             Student** outputStudents = new Student*[size];
             size_t outputSize = 0;
```

## Продолжение приложения А

```
          double minScholarship = INT_MAX;
             double maxScholarship = INT_MIN;
             for (size_t j = 0; j < size; j++) {
                 if (strcmp(students[j]->getFaculty(),
faculties[i]) == 0) {
                     outputStudents[outputSize++] = students[j];

                     minScholarship = (minScholarship >
students[j]->getScholarshipAmount() ?
                         students[j]->getScholarshipAmount() :
minScholarship);
                     maxScholarship = (maxScholarship <
students[j]->getScholarshipAmount() ?
                          students[j]->getScholarshipAmount() :
maxScholarship);
                 }
             }


FileManager::writeScholarshipStatisticsToFile(resultFilename,
outputStudents, outputSize,
                 faculties[i], minScholarship, maxScholarship);

             delete[] outputStudents;
         }

         AdditionalFunctions::deleteStudentsArray(students,
size);
     }


     main.cpp
     #include <iostream>
     #include "Database.h"
     #include "FileManager.h"
     #include "Operations.h"

     int main() {

         const char* storageFilename = "Storage.bin";
         const char* resultFilename = "result.txt";
         size_t charSize = 255;

         int choose = -1;
         while (choose != 0){
             std::cout << "----- Instruction -----";
             std::cout << '\n' << "0. Quit program";
             std::cout << '\n' << "1. Database operations";
             std::cout << '\n' << "2. Search";
             std::cout << '\n' << "3. Sorting";
             std::cout << '\n' << "4. Statistics";
```

## Продолжение приложения А

```
            std::cout << '\n' << "Choose an option: ";
            std::cin >> choose;

            switch (choose) {
                case 1: {
                    std::cout << "-----------------------" <<
std::endl;
                    std::cout << "0. Return to main
instructions" << std::endl;
                    std::cout << "1. Display all students" <<
std::endl;
                    std::cout << "2. Add new student" <<
std::endl;
                    std::cout << "3. Delete student" <<
std::endl;
                    std::cout << "4. Clear database" <<
std::endl;
                    std::cout << "5. Edit student info" <<
std::endl;
                    std::cout << "Choose an option: ";
                    int choose2;
                    std::cin >> choose2;
                    switch (choose2) {
                        case 1: {

Database::displayAll(storageFilename);
                            break;
                        }
                        case 2: {
                            std::cin.ignore();

                            std::cout << "Enter student's full
name: ";
                            char* fullName = new char[charSize];
                            std::cin.getline(fullName,
charSize);

                            std::cout << "Enter student's
faculty: ";
                            char* faculty = new char[charSize];
                            std::cin.getline(faculty, charSize);

                            std::cout << "Enter student's grade:
";
                            char* grade = new char[charSize];
                            std::cin.getline(grade, charSize);
                            double averageGrade;
                            if
(AdditionalFunctions::isDigit(grade))
                                averageGrade = std::stof(grade);
                            else {
```

## Продолжение приложения А

```
                                    std::cout << "Invalid input" <<
std::endl;
                                    break;
                                }

                                std::cout << "Enter student's
privileges (1 or 0): ";
                                char* privileges = new
char[charSize];
                                std::cin.getline(privileges,
charSize);
                                bool hasPrivileges = false;
                                if (strcmp(privileges, "1") == 0)
                                    hasPrivileges = true;
                                else if (strcmp(privileges, "0") ==
0)
                                    hasPrivileges = false;
                                else {
                                    std::cout << "Invalid input" <<
std::endl;
                                    break;
                                }

                                std::cout << "Enter student's
scholarship: ";
                                char* scholarship = new
char[charSize];
                                std::cin.getline(scholarship,
charSize);
                                double scholarshipAmount;
                                if
(AdditionalFunctions::isDigit(scholarship))
                                    scholarshipAmount =
std::stof(scholarship);
                                else {
                                    std::cout << "Invalid input" <<
std::endl;
                                    break;
                                }


Database::addStudent(storageFilename, new Student(fullName,
faculty,
                                    averageGrade, hasPrivileges,
scholarshipAmount));

                                delete[] privileges;
                                delete[] scholarship;
                                delete[] grade;
                                delete[] faculty;
                                 delete[] fullName;
```

## Продолжение приложения А

```
                            break;
                        }
                        case 3: {
                            size_t studentNumber;
                            size_t cnt =
FileManager::getNumberOfStudentsInFile(storageFilename);
                            std::cout << "Enter student number
to delete (1-" << cnt << "): ";
                            std::cin >> studentNumber;

                            if
(Database::deleteStudent(storageFilename, studentNumber)) {
                                std::cout << "Student deleted
successfully.\n";
                            } else {
                                std::cerr << "Failed to delete
student.\n";
                            }
                            break;
                        }
                        case 4: {
                            std::cout << "Confirmation (1-yes,
0-no): ";
                            std::cin.ignore();
                            char isCorrect[charSize];
                            std::cin.getline(isCorrect,
charSize);

                            if (strcmp(isCorrect, "1") != 0) {
                                std::cout << "Aborting
operation...\n";
                                break;
                            }


FileManager::clearFile(storageFilename);
                            break;
                        }
                        case 5: {
                            Student** students = nullptr;
                            size_t count = 0;

                            if (!
FileManager::readStudentsFromFile(storageFilename, students,
count)) {
                                std::cerr << "Database is empty
or cannot be accessed.\n";
                                break;
                            }
      int studentNumber;
```

## Продолжение приложения А

```
                                std::cout << "Enter student number
to edit (1-" << count << "): ";
                                std::cin >> studentNumber;

                                if (studentNumber < 1 ||
studentNumber > count) {
                                    std::cerr << "Invalid student
number.\n";

AdditionalFunctions::deleteStudentsArray(students, count);
                                    break;
                                }

                                std::cin.ignore(); // Consume
newline character

                                Student* oldStudent =
students[studentNumber - 1];

                                char    fullName[charSize],
faculty[charSize],
                                        gradeStr[charSize],
privStr[charSize],
                                        scholStr[charSize];

                                std::cout << "Current full name: "
<< oldStudent->getFullName()
                                          << ". Enter new (press
Enter to keep): ";
                                std::cin.getline(fullName,
charSize);

                                std::cout << "Current faculty: " <<
oldStudent->getFaculty()
                                          << ". Enter new (press
Enter to keep): ";
                                std::cin.getline(faculty, charSize);

                                std::cout << "Current average grade:
" << oldStudent->getAverageGrade()
                                          << ". Enter new (press
Enter to keep): ";
                                std::cin.getline(gradeStr,
charSize);

                                std::cout << "Current privileges (0
or 1): " << oldStudent->getHasPrivileges()
                                          << ". Enter new (press
Enter to keep): ";
                                std::cin.getline(privStr, charSize);
```

## Продолжение приложения А

```
                                std::cout << "Current scholarship
amount: " << oldStudent->getScholarshipAmount()
                                          << ". Enter new (press
Enter to keep): ";
                                std::cin.getline(scholStr,
charSize);

                                // Validate and build new values
                                const char* newFullName =
strlen(fullName) > 0 ? fullName : oldStudent->getFullName();
                                const char* newFaculty =
strlen(faculty) > 0 ? faculty : oldStudent->getFaculty();

                                double newGrade = oldStudent-
>getAverageGrade();
                                if (strlen(gradeStr) > 0) {
                                    if (!
AdditionalFunctions::isDigit(gradeStr)) {
                                        std::cerr << "Invalid grade
input.\n";

AdditionalFunctions::deleteStudentsArray(students, count);
                                        break;
                                    }
                                    newGrade = std::stof(gradeStr);
                                }

                                bool newPrivilege = oldStudent-
>getHasPrivileges();
                                if (strlen(privStr) > 0) {
                                    if (strcmp(privStr, "0") == 0)
                                        newPrivilege = false;
                                    else if (strcmp(privStr, "1") ==
0)
                                        newPrivilege = true;
                                    else {
                                        std::cerr << "Invalid
privileges input.\n";

AdditionalFunctions::deleteStudentsArray(students, count);
                                        break;
                                    }
                                }

                                double newScholarship = oldStudent-
>getScholarshipAmount();
                                if (strlen(scholStr) > 0) {
                                    if (!
AdditionalFunctions::isDigit(scholStr)) {
      std::cerr << "Invalid scholarship input.\n";
```

## Продолжение приложения А

```
AdditionalFunctions::deleteStudentsArray(students, count);
                                    break;
                                }
                                newScholarship =
std::stof(scholStr);
                            }

                            // Create updated student
                            Student* updatedStudent = new
Student(
                                const_cast<char*>(newFullName),
                                const_cast<char*>(newFaculty),
                                newGrade,
                                newPrivilege,
                                newScholarship
                            );

                            std::cout << "Check correct input:
\n";
                            updatedStudent->print();
                            std::cout << "Confirmation of
changes to records (1 – yes, 0 – no): ";
                            char isCorrectInput[charSize];
                            std::cin.getline(isCorrectInput,
charSize);

                            if (strcmp(isCorrectInput, "1") !=
0) {
                                std::cerr << "Aborting
operation...\n";
                                delete updatedStudent;

AdditionalFunctions::deleteStudentsArray(students, count);

                                break;
                            }

                            // Perform the update
                            if (!
Database::changeStudentInfo(storageFilename, updatedStudent,
studentNumber)) {
                                std::cerr << "Failed to update
student information.\n";
                                delete updatedStudent;
                            }


AdditionalFunctions::deleteStudentsArray(students, count);
                            break;
                        }
```

## Продолжение приложения А

```
                        case 0: {
                            break;
                        }
                        default: {
                            std::cout << "Wrong choose.
Returning to main instructions" << std::endl;
                            break;
                        }
                    }

                    break;
                }
                case 2: {
                    std::cout << "----------------------" <<
std::endl;
                    std::cout << "0. Return to main
instructions" << std::endl;
                    std::cout << "1. Linear search by name" <<
std::endl;
                    std::cout << "2. Binary search by grade" <<
std::endl;
                    std::cout << "Choose an option: ";
                    int choose2;
                    std::cin >> choose2;

                    switch (choose2) {
                        case 1: {
                            char* name = new char[charSize];
                            std::cout << "Enter student's full
name: ";
                            std::cin.ignore();
                            std::cin.getline(name, charSize);

Database::linearSearchByName(storageFilename, name);
                            break;
                        }
                        case 2: {
                            double grade;
                            std::cout << "Enter student's grade:
";
                            std::cin >> grade;

Database::binarySearchByGrade(storageFilename, grade);
                            break;
                        }
                        case 0: {
                            break;
                        }
                        default: {
                            std::cout << "Wrong choose.
Returning to main instructions" << std::endl;
```

## Продолжение приложения А

```
                            break;
                        }
                    }
                    break;
                }
                case 3: {
                    std::cout << "----------------------" <<
std::endl;
                    std::cout << "0. Return to main
instructions" << std::endl;
                    std::cout << "1. Quick sort by scholarship"
<< std::endl;
                    std::cout << "2. Selection sort by grade" <<
std::endl;
                    std::cout << "3. Insertion sort by name" <<
std::endl;
                    std::cout << "Choose an option: ";
                    int choose2;
                    std::cin >> choose2;

                    switch (choose2) {
                        case 1: {

Database::quickSortByScholarship(storageFilename);
                            break;
                        }
                        case 2: {

Database::selectionSortByGrade(storageFilename);
                            break;
                        }
                        case 3: {

Database::insertionSortByName(storageFilename);
                            break;
                        }
                        case 0: {
                            break;
                        }
                        default: {
                            std::cout << "Wrong choose.
Returning to main instructions" << std::endl;
                            break;
                        }
                    }
                    break;
                }
                case 4: {
                    std::cout << "----------------------" <<
std::endl;
```

## Продолжение приложения А

```
                    std::cout << "0. Return to main
instructions" << std::endl;
                    std::cout << "1. Identify students for an
increased scholarship" << std::endl;
                    std::cout << "2. Scholarship statistics" <<
std::endl;
                    std::cout << "Choose an option: ";
                    int choose2;
                    std::cin >> choose2;

                    switch (choose2) {
                        case 1: {
                            double grade = 0;
                            std::cout << "Enter student's grade:
";
                            std::cin >> grade;

                            bool privileges = false;
                            std::cout << "Enter student's
privileges (1 or 0): ";
                            std::cin >> privileges;


Database::identifyIncreasedScholarship(storageFilename,
                                    resultFilename, grade,
privileges);

                            break;
                        }
                        case 2: {

Database::scholarshipStatistics(storageFilename,
resultFilename);
                            break;
                        }
                        case 0: {
                            break;
                        }
                        default: {
                            std::cout << "Wrong choose.
Returning to main instructions" << std::endl;
                            break;
                        }
                    }
                    break;
                }
                case 0: {
                    std::cout << "Program exiting..." <<
std::endl;

                    break;
```

## Продолжение приложения А

```
                }
                default: {
                    std::cerr << "Wrong choice." << std::endl;
                    continue;
                }
            }
        }

        std::cout << "The program has been successfully
completed!" << std::endl;

        return 0;
    }
```

# ПРИЛОЖЕНИЕ Б
# (обязательное)
# Блок–схема работы программы

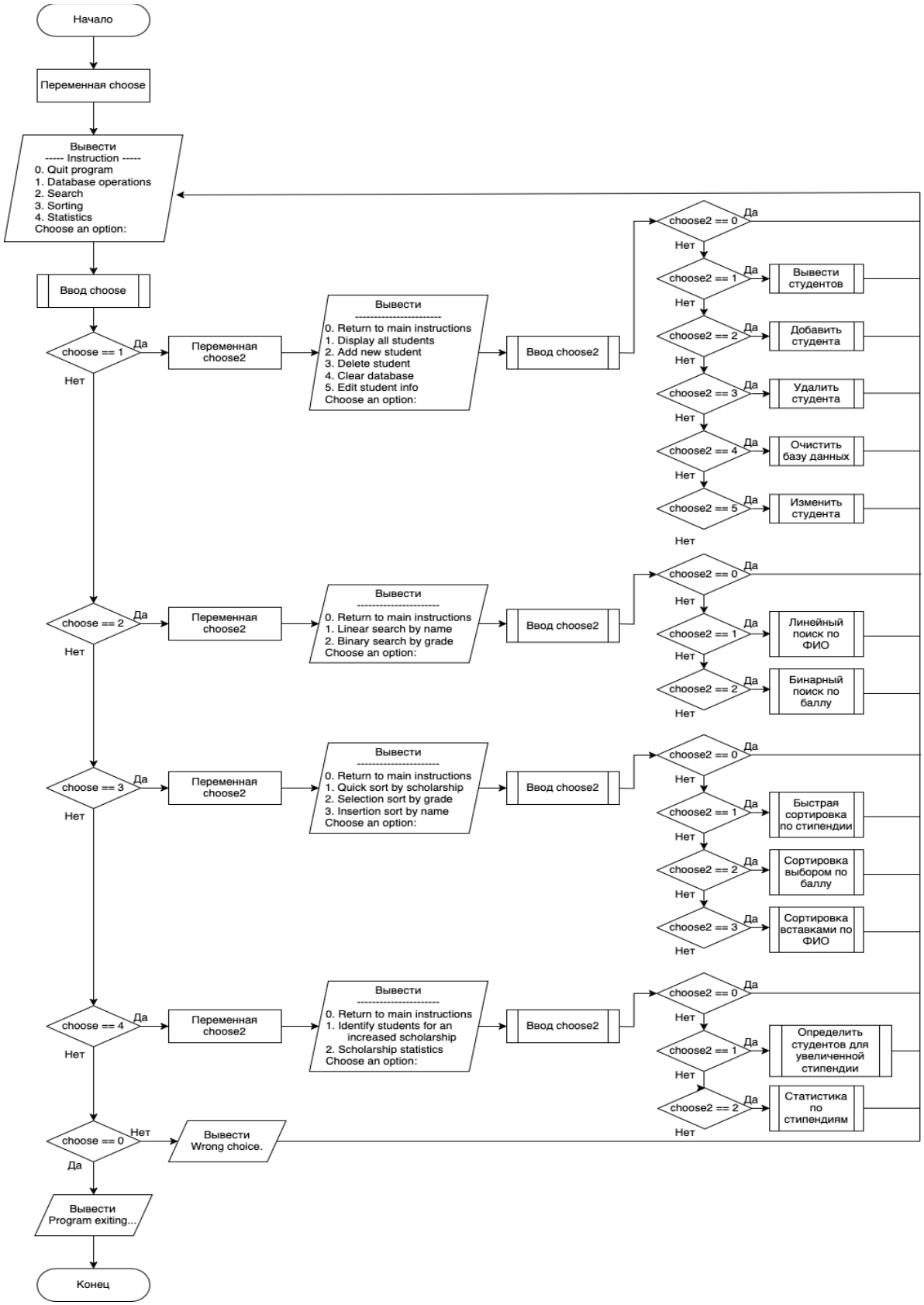


Рисунок Б.1 – Блок–схема работы программы

# ВЕДОМОСТЬ ДОКУМЕНТОВ

| Обозначение | Наименование | Дополнительные сведения |
|---|---|---|
| | | |
| | *Текстовые документы* | |
| | | |
| *БГУИР КП 6–05–0611–03 033 ПЗ* | *Пояснительная записка* | *64 с.* |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ***БГУИР КП 6–05–0611–03 033 ВД*** | | |
| | | | | | | | |
| | | | | | | | |
| *Изм.* | *Л.* | *№ докум.* | *Подп.* | *Дата* | *Система расчета стипендии обучающимся* | *Лит* | *Лист* / *Листов* |
| *Разраб.* | | *Захаренков* | | | | *У* | *64* / *64* |
| *Проверил* | | *Панасик* | | | | | |
| | | | | | | *Кафедра ВМиП гр. 421702* | |
| | | | | | | | |
| | | | | | | | |