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 курсовому проекту

на тему:

**«Система учета реализации проездных билетов»**

Выполнил: гр.421702 Сушко В.Р.

Проверила: Панасик А.А.

Минск 2025

**РЕФЕРАТ**

СИСТЕМА УЧЕТА РЕАЛИЗАЦИИ ПРОЕЗДНЫХ БИЛЕТОВ: курсовой проект/Сушко В.Р. Минск БГУИР ,2025

В данной работе рассматривается разработка программного средства для автоматизации системы реализации проездных билетов.Основная цель проекта – создание удобной и эффективной программы, которая позволяет выполнять основные операции по обработке бинарных файлов в консольном режиме,существенно облегчая анализ данных о проездных билетах.

Проект реализует стандартные операции обработки файлов данных:  
создание нового файла,просмотр,добавление,редактирование и удаление записей. Также реализованы алгоритмы линейного поиска и бинарного поиска, быстрой сортировки,сортировки вставками,сортировки выбором.

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

**СОДЕРЖАНИЕ**

[Введение 7](#_Toc193324194)

[1 Структуры и файлы 9](#_Toc193324195)

[1.1 Используемые структеры данных 9](#_Toc193324196)

[1.2 Работа с файлами 10](#_Toc193324196)

[1.3 Вывод 10](#_Toc193324196)

[2 Алгоритмы сортировки 12](#_Toc193324197)

[2.1 Быстрая сортировка 12](#_Toc193324196)

[2.2 Сортировка Вставками 14](#_Toc193324196)

[2.3 Сортировка Выбором 14](#_Toc193324196)

[2.4 Сравнение методов сортировки 15](#_Toc193324196)

[2.5 Особые случаи сортировки 15](#_Toc193324196)

[2.6 Вывод 16](#_Toc193324196)

[3 Алгоритмы поиска 17](#_Toc193324198)

[3.1 Линейный поиск 17](#_Toc193324196)

[3.2 Бинарный поиск 17](#_Toc193324196)

[3.3 Особые случаи 18](#_Toc193324196)

[3.4 Вывод 18](#_Toc193324196)

[4 Пользовательские функции 20](#_Toc193324199)

[4.1 Ввод-вывод данных 20](#_Toc193324196)

[4.2 Поиск данных 20](#_Toc193324196)

[4.3 Алгоритмы сортировки 21](#_Toc193324196)

[4.4 Всомогательные функции 21](#_Toc193324196)

[4.5 Специализированные функции 22](#_Toc193324196)

[4.6 Ключевые особенности реализации 22](#_Toc193324196)

[4.7 Заключение 22](#_Toc193324196)

[5 Описание работы программы 24](#_Toc193324200)

[5.1 Запуск программы 24](#_Toc193324196)

[5.2 Основные операции 24](#_Toc193324196)

[5.3 Поиск и сортировка 25](#_Toc193324196)

[5.4 Особые ситуации 26](#_Toc193324196)

[5.5 Рекомендации по работе 26](#_Toc193324196)

[5.6 Типовые сценарии 27](#_Toc193324196)

[5.7 Вывод 27](#_Toc193324196)

[Заключение 28](#_Toc193324201)

[Список использованных источников 29](#_Toc193324202)

[Приложение А (обязательное) Листинг кода 30](#_Toc193324203)

[Приложение Б (обязательное) Блок-схема работы программы 46](#_Toc193324204)

Введение

Современные системы управления транспортными перевозками требуют эффективных инструментов для хранения, обработки и анализа данных. Железнодорожные рейсы являются одним из ключевых элементов транспортной инфраструктуры, и автоматизация работы с их расписанием, ценами и доступностью позволяет оптимизировать логистику, улучшить обслуживание пассажиров и снизить нагрузку на операторов.

Объектом исследования данной работы является программное обеспечение для управления данными о железнодорожных рейсах. Программа предоставляет функционал для добавления, поиска, сортировки и статистического анализа рейсов, что делает её полезной как для пассажиров, так и для сотрудников транспортных компаний.

Цель работы – разработка консольного приложения на языке C++, позволяющего:

* хранить информацию о рейсах (название, направление, цена, номер, дата и время);
* осуществлять быстрый поиск по различным критериям;
* сортировать данные для удобного представления;
* анализировать статистику за указанный период.

Для достижения поставленной цели необходимо решить следующие задачи:

* Разработать структуру данных для хранения информации о рейсах.
* Реализовать функции ввода и вывода данных, в том числе с использованием файлов.
* Обеспечить возможность поиска рейсов по городам, названию, номеру и другим параметрам.
* Реализовать алгоритмы сортировки (быстрая сортировка, сортировка вставками, выбором) для удобного представления данных.
* Добавить функцию формирования статистики за выбранный период.

Актуальность темы обусловлена возрастающим спросом на цифровизацию транспортных систем. Автоматизация учета рейсов позволяет:

* сократить время обработки запросов пассажиров;
* минимизировать ошибки, связанные с ручным вводом данных;
* упростить анализ загруженности направлений.

Разработанная программа может быть использована в кассах железнодорожных вокзалов, туристических агентствах или интегрирована в более сложные системы управления перевозками.

В дальнейшем работа может быть расширена за счет добавления графического интерфейса, поддержки сетевого взаимодействия или подключения базы данных для хранения информации.

1 Структуры и файлы

1.1. Используемые структуры данных

Основной структурой данных в программе является Train, которая хранит информацию о железнодорожных рейсах.

struct Train {

double price[256]; // Цена билета

int number[256]; // Номер рейса

char start\_town[256][256]; // Город отправления

char finish\_town[256][256]; // Город прибытия

char name[256][256]; // ФИО

char date[256][256]; // Дата в формате "DD.MM.YYYY"

char time[256][256]; // Время в формате "HH:MM"

int n; // Текущее количество записей

};

Для хранения информации о железнодорожных рейсах была разработана структура данных, использующая несколько типов массивов, каждый из которых оптимально подходит для хранения конкретного типа информации. Основой хранения числовых данных стали массивы типа double и int. Массив price[256] типа double был выбран для хранения стоимости билетов, поскольку этот тип данных позволяет точно работать с дробными значениями, что особенно важно при отображении денежных сумм (например, 1250.50 рублей). Для хранения номеров рейсов используется массив number[256] типа int, так как номера рейсов по своей природе являются целыми числами.

Текстовые данные организованы с использованием двумерных массивов символов (char[]). Для хранения названий городов отправления и назначения используются массивы start\_town[256][256] и finish\_town[256][256] соответственно. Аналогичный подход применен для хранения ФИО пассажиров в массиве name[256][256]. Такой выбор обусловлен необходимостью обеспечения совместимости с функциями файлового ввода-вывода стандартной библиотеки C (такими как fgets и fputs), а также упрощением процесса сериализации данных при сохранении в файл.

Особое внимание уделено хранению временных данных. Как отмечает Р. Лафоре [1, с. 342], выбор формата хранения даты и времени является критически важным для последующей обработки временных данных. Даты рейсов хранятся в массиве date[256][256] в формате "DD.MM.YYYY" (например, "12.05.2024"), а время - в массиве time[256][256] в формате "HH:MM" (например, "14:30"). Раздельное хранение даты и времени было сознательным решением, так как это значительно упрощает операции сравнения и сортировки временных данных. Для отслеживания количества загруженных записей используется переменная-счетчик n.

Выбранный формат хранения данных имеет несколько важных преимуществ. Использование фиксированных массивов размером [256] значительно упрощает работу с файлами, обеспечивая предсказуемость размера структур данных. Однако это накладывает ограничение на максимальное количество обрабатываемых записей. Применение C-строк (char[]) вместо более современных альтернатив обусловлено требованиями совместимости с файловыми операциями и необходимостью работы в различных средах выполнения. Организация данных в виде набора отдельных полей, каждое из которых отвечает за конкретный аспект информации о рейсе, обеспечивает четкую структуру и облегчает реализацию алгоритмов обработки данных.

**1.2. Работа с файлами**

Программа поддерживает чтение и запись данных в текстовые файлы

Формат хранения данных в файле:

Каждый рейс записывается в виде 7 строк (по одной на каждое поле):

* ФИО
* Город отправления
* Город прибытия
* Цена
* Номер рейса
* Время
* Дата

Функции для работы с файлами:

* read(const char\* inputFile, int& i)` – загружает данные из файла в структуру.
* writeFile(const char\* file) – сохраняет все рейсы в файл.
* CreateFile(char\* file) – создает пустой файл (используется для очистки).
* StatistickaFile() – записывает статистику за период в указанный файл.

1.3. Вывод

В рамках данного проекта была реализована структура Train, основанная на использовании фиксированных массивов. Такой подход значительно упрощает работу с файлами, обеспечивая удобство при записи и обработке данных. Однако применение фиксированных массивов накладывает определенные ограничения на максимальное количество записей, что необходимо учитывать при масштабировании и расширении функциональных возможностей системы.

Одной из ключевых особенностей проекта является хранение данных в текстовых файлах с четко установленным форматом. Это решение обеспечивает прозрачность структуры данных и удобство их обработки. Формат файлов заранее определен таким образом, чтобы сохраняемая информация была легко доступна и могла быть изменена вручную при необходимости. Такой способ организации данных позволяет пользователю не только считывать информацию, но и вносить корректировки без необходимости использования специализированного программного обеспечения.

При хранении данных особое внимание уделено их структурированности, что позволяет минимизировать вероятность возникновения ошибок при обработке информации. Использование фиксированных массивов в структуре Train гарантирует стабильность работы системы и предсказуемость результатов, что особенно важно при обработке больших объемов данных.

Таким образом, выбранная методология организации данных обеспечивает надежность, удобство использования и простоту редактирования. Несмотря на ограничение в максимальном количестве записей, структура Train остается эффективным инструментом для работы с транспортными расписаниями, позволяя пользователям получать и модифицировать информацию в удобном формате. Такой подход способствует упрощению взаимодействия с данными и повышению производительности системы в рамках поставленных задач.

2 Алгоритмы сортировки

2.1. Быстрая сортировка

В программе реализовано три алгоритма сортировки, каждый из которых применяется для определенных типов данных и решает конкретные задачи по упорядочиванию информации о железнодорожных рейсах.

Алгоритм быстрой сортировки (quicksort) начинается с выбора опорного элемента (pivot), который играет ключевую роль в процессе сортировки. В классической реализации в качестве опорного элемента обычно выбирается последний элемент массива, однако возможны и другие стратегии: случайный выбор (для уменьшения вероятности наихудшего случая), медиана трёх (первого, среднего и последнего элементов) или фиксированный элемент (например, первый или средний). Выбор оптимального опорного элемента существенно влияет на эффективность алгоритма, особенно при работе с частично упорядоченными данными.

После выбора опорного элемента выполняется разбиение массива (partitioning), в ходе которого массив разделяется на две части относительно этого элемента. Левая часть содержит элементы, меньшие либо равные опорному, а правая – элементы, большие опорного. Этот этап выполняется за линейное время (O(n)) с использованием метода двух указателей (метода Хоара). Один указатель последовательно перемещается от начала массива, сравнивая элементы с опорным, а второй фиксирует границу между меньшими и большими элементами. При обнаружении элемента, меньшего опорного, происходит обмен, и граница сдвигается. В результате опорный элемент занимает свою окончательную позицию в отсортированном массиве.

Затем алгоритм рекурсивно применяется к обеим частям массива: сортируется подмассив элементов, меньших опорного, и подмассив элементов, больших опорного. Рекурсия продолжается до тех пор, пока размер подмассива не станет равным 1, что является базовым случаем рекурсии, так как массив из одного элемента считается уже отсортированным. Таким образом, комбинация выбора опорного элемента, эффективного разбиения и рекурсивной обработки подмассивов обеспечивает высокую производительность алгоритма в среднем случае.

Заключение

Быстрая сортировка является одним из наиболее эффективных алгоритмов сортировки со средней временной сложностью O(n log n). Однако в худшем случае (например, при неудачном выборе опорного элемента и уже отсортированном массиве) сложность может достигать O(n²). Для оптимизации алгоритма применяются различные модификации, включая рандомизированный выбор опорного элемента и гибридные схемы (например, переход на сортировку вставками для небольших подмассивов).

Рекурсивная реализация QuickSort:

void quickSort(int low, int high) {

if (low < high) { элемента)

int pi = partition(low, high);

quickSort(low, pi - 1);

quickSort(pi + 1, high);

}

}

Анализ сложности:

* Средний случай: O(n log n) – достигается при равномерном разбиении массива.
* Худший случай:O(n²) – возникает, если опорный элемент всегда оказывается минимальным или максимальным (например, при сортировке уже упорядоченного массива с выбором крайнего элемента в качестве pivot).

Оптимизации:

* Рандомизированный выбор опорного элемента – уменьшает вероятность худшего случая.
* Использование сортировки вставками для небольших подмассивов (обычно при n < 10).
* Медиана трёх (первый, средний, последний элементы) в качестве pivot.

Специфика сортировки дат и времени: Для корректного упорядочивания записей с датами ("DD.MM.YYYY") и временем ("HH:MM") применяется специальная функция сравнения dated(), которая выполняет последовательное сравнение компонентов:

* Сравнение годов (`YYYY`) – более поздняя дата считается больше.
* Если годы равны → сравнение месяцев (MM)
* Если месяцы равны → сравнение дней (DD)
* Если даты идентичны → сравнение времени:
* Сначала часы (HH), затем минуты (MM).

Пример работы:

* 01.01.2023 < 02.01.2023 (сравнение по дню).
* 15.05.2023 12:30 > 15.05.2023 10:15` (одинаковые даты, сравнение по времени).

Таким образом, сочетание быстрой сортировки и правильного сравнения дат обеспечивает оптимальную обработку временных данных.

2.2 Сортировка выбором (Selection Sort)

Сложность: Всегда O(n²), так как алгоритм выполняет полный проход по неотсортированной части на каждой итерации, независимо от исходного порядка данных.

Особенности:

* Не адаптивный – время работы не улучшается на частично упорядоченных данных.
* Минимум обменов – за n итераций выполняется только n перестановок (полезно, если обмены дорогие).
* Неустойчивый – может менять порядок элементов с одинаковыми значениями.

2.3. Сортировка вставками (Insertion Sort)

Сложность:

* Худший случай:O(n²) – если массив отсортирован в обратном порядке.
* Лучший случай: O(n) – если массив уже отсортирован (внутренний цикл не выполняется).
* Средний случай: O(n²) – но эффективен для небольших n или частично упорядоченных данных.

Особенности:

* Адаптивный – работает быстрее на почти упорядоченных данных.
* Устойчивый – не меняет порядок равных элементов.
* Эффективен для небольших массивов (n < 50) или в комбинации с другими алгоритмами (например, в QuickSort для маленьких подмассивов).

Специфика алфавитной сортировки: Функция String() реализует лексикографическое сравнение строк по правилам:

* Посимвольное сравнение ASCII-кодов: "A" (65) < "B" (66)`, `"a" (97) > "A" (65).
* Учет длины строк: Если символы совпадают, более короткая строка считается меньшей (`"abc" < "abcd"`).
* Регистрозависимость: Заглавные и строчные буквы сравниваются по их ASCII-кодам (`"Apple" < "apple"`).

Пример работы:

* Berlin < Paris` (сравнение по B и P ).
* Moscow > Madrid (первые символы равны M, но o > a).
* Rome < Romeo (одинаковое начало, но вторая строка длиннее).

Вывод:

* Сортировка выбором – проста, но неэффективна для больших данных.
* Сортировка вставками – предпочтительна для небольших или частично упорядоченных наборов.
* Лексикографическое сравнение требует аккуратной реализации для корректной работы с Unicode и регистром.

**2.4. Сравнение методов сортировки**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Лучший случай | Худший случай | Память | Устройчивость | Применение в проекте |
| Быстрая сортировка | O(n log n) | O(n²) | O(log n) | Неустойчивая | Сортировка по дате |
| Сортировка Вставками | O(n) | O(n²) | O(1) | Устойчивая | Сортировка по городу |
| Сортировка выбором | O(n²) | O(n²) | O(1) | Неустойчивая | Сортировка по цене |

Таблица 2.1 – Таблица сравнения алгоритмов сортировки,которые использовались в проекте

**2.5. Особые случаи сортировки**

* Сортировка текстовых данных:
* Учитывается регистр символов
* Строки разной длины сравниваются корректно
* Русские и английские символы сортируются согласно их ASCII-кодам

Сортировка по составным полям: При равных датах выполняется дополнительное сравнение по времени:

if(Double(d1) == Double(d2)) {

* + return Double(h1) < Double(h2);

}

Сортировка чисел с плавающей точкой: Цены сравниваются как числа double, что требует особой внимательности при работе с денежными значениями.

**2.6 Вывод**

Реализованные алгоритмы сортировки – быстрая сортировка, сортировка выбором и сортировка вставками – обеспечивают программе мощный инструментарий для гибкого упорядочивания данных по различным критериям. Каждый из этих методов обладает уникальными характеристиками, что позволяет оптимально решать разнообразные задачи в зависимости от типа данных и требуемой скорости обработки. Быстрая сортировка демонстрирует выдающуюся производительность на больших массивах, сортировка выбором эффективна при работе с небольшими наборами данных или когда критично количество перезаписей, а сортировка вставками идеально подходит для почти упорядоченных данных или случаев, когда важна стабильность алгоритма.

Особое внимание уделено обработке сложных случаев: корректному сравнению текстовых данных с учетом регистра и длины строк, сортировке составных полей (например, дата и время), а также точному сравнению чисел с плавающей точкой, что особенно важно при работе с финансовыми показателями. Программа учитывает различные нюансы, включая обработку специальных значений вроде NaN, сравнение строк с кириллицей и латиницей, а также каскадное сравнение дат и времени.

Перспективы развития системы включают внедрение более совершенных алгоритмов вроде TimSort, добавление параллельных вычислений для ускорения обработки больших объемов данных, улучшение поддержки локализации и реализацию интерактивной сортировки. Эти улучшения позволят добиться производительности, сопоставимой с промышленными системами управления базами данных. Главный принцип, заложенный в архитектуру программы – адаптивный выбор оптимального алгоритма под конкретную задачу – гарантирует высокую эффективность работы при любых условиях и типах данных, обеспечивая тем самым надежность и масштабируемость решения.

3 Алгоритмы поиска

**3.1. Линейный поиск**

Применение:

* Поиск по названию рейса (SearchByName)
* Поиск по городу отправления (SearchByOnlyStartTown)
* Поиск по городу прибытия (SearchByOnlyFinishTown)

Поиск по направлению (город отправления + город прибытия, SearchByStartTown)

Как отмечает Р. Лафоре [1], линейный поиск является базовым алгоритмом обработки данных. В нашей программе он реализован следующим образом:

void SearchByName(int start, char\* s1) {

bool flag = true;

for(int i = start; i < n; i++) {

for(int j = 0; s1[j] != '\0'; j++) {

if(name[i][j] != s1[j]) {

flag = false;

}

}

if(flag) {

print(i);

}

flag = true;

}

}

Особенности:

* Последовательное сравнение каждого элемента
* Поддержка частичного совпадения строк
* Сложность O(n), где n - количество записей
* Чувствительность к регистру символов

Преимущества:

* Простота реализации
* Работает с неотсортированными данными
* Универсальность (можно искать по любому полю)

**3.2. Бинарный поиск**

Условия работы:

* Требуется предварительная сортировка данных (sortBinary)
* Работает только с числовыми полями (номер рейса)
* Сложность O(log n)

Процесс поиска:

* Сравнение с элементом в середине массива
* Если искомое значение меньше - поиск в левой части
* Если больше - в правой части
* Рекурсивное повторение до нахождения элемента

3.3. Особые случаи поиска

Поиск по составному условию:

void SearchByStartTown(char\* s1, char\* s2) {

// Поиск по двум городам одновременно

// (отправление и прибытие)

}

Поиск по дате:

Используется функция dated() для корректного сравнения дат в формате "DD.MM.YYYY"

Поиск с фильтрацией:

void Statisticka(char\* date\_1, char\* date\_2) {

// Поиск всех рейсов в заданном временном промежутке

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Лучший случай | Худший случай | Требование к данным | Применение в проекте |
| Линейный поиск | O(1) | O(n) | Нет требований | Поиск по названию, городам |
| Бинарный поиск | O(1) | O(log n) | Отсортированный массив | Поиск по номеру рейса |

Таблица 3.1 – Таблица сравнения алгоритмов поиска,которые использовались в проекте

**3.4 Вывод**

Реализованные в системе алгоритмы поиска предоставляют гибкий и эффективный механизм доступа к информации о рейсах, позволяя оптимально решать различные задачи в зависимости от характера запросов и требований к производительности. Линейный поиск служит универсальным инструментом, обеспечивающим работу с данными любого типа без необходимости их предварительной подготовки, что особенно ценно при обработке неструктурированной информации или выполнении единичных запросов. В тех случаях, когда критична скорость выполнения операций и имеется возможность предварительной сортировки числовых данных, бинарный поиск демонстрирует исключительную эффективность, значительно сокращая время доступа к информации за счет логарифмической сложности алгоритма.

Особое значение имеет корректный выбор метода поиска в зависимости от конкретной ситуации: при работе с небольшими объемами данных или при необходимости частого обновления информации предпочтение отдается линейному поиску благодаря его простоте и отсутствию требований к поддержанию определенного порядка данных. В то же время для крупных отсортированных массивов, особенно при интенсивных запросах на чтение, бинарный поиск становится незаменимым инструментом, обеспечивающим максимальную производительность.

Перспективы развития системы поиска включают внедрение более сложных алгоритмов (таких как интерполяционный поиск для равномерно распределенных данных), реализацию хеширования для мгновенного доступа к часто запрашиваемым элементам, а также оптимизацию работы с составными ключами поиска. Важным направлением совершенствования является также разработка гибридных подходов, автоматически выбирающих оптимальный алгоритм поиска на основе анализа характеристик обрабатываемых данных и статистики запросов. Такой интеллектуальный подход позволит системе динамически адаптироваться к изменяющимся условиям работы, обеспечивая стабильно высокую производительность при минимальных затратах вычислительных ресурсов.

Грамотное сочетание различных методов поиска в рамках единой системы создает надежную основу для эффективной работы с данными о рейсах, гарантируя как оперативность обработки запросов, так и гибкость при работе с разнородной информацией. Это особенно важно в условиях, когда от скорости и точности поиска напрямую зависит качество обслуживания клиентов и эффективность управления транспортными потоками.

4 Пользовательские функции

4.1 Ввод-вывод данных

Реализованные механизмы ввода-вывода обеспечивают надежное взаимодействие с файловой системой. Функция read выполняет загрузку данных из указанного файла, строго соблюдая заданный формат представления информации. Особое внимание уделено автоматическому контролю количества загруженных записей через механизм передачи параметра по ссылке, что позволяет синхронно отслеживать объем обрабатываемых данных. При обработке каждой строки файла осуществляется проверка соответствия установленному формату, гарантируя целостность и корректность загружаемой информации. Функция writeFile обеспечивает сохранение всего массива данных в файл, используя тот же формат, что и при чтении. Особенностью реализации является полная перезапись содержимого целевого файла, что исключает возможность частичного сохранения или дублирования информации. После успешного завершения операции сохранения система выводит соответствующее уведомление, позволяя пользователю убедиться в корректности выполнения операции.

4.2 Поиск данных

Система предоставляет два принципиально разных подхода к поиску информации, каждый из которых оптимизирован для конкретных сценариев работы. Функция SearchByStartTown реализует текстовый поиск по точному совпадению, что особенно востребовано при работе с названиями населенных пунктов. Регистрозависимый алгоритм сравнения обеспечивает высокую точность результатов, а мгновенный вывод найденных записей позволяет пользователю оперативно получать необходимую информацию. Для числовых данных, в частности номеров рейсов, предусмотрена функция binary\_search, использующая алгоритм бинарного поиска. Важной особенностью является требование предварительной сортировки данных, что компенсируется максимальной скоростью поиска в отсортированном массиве. В случае отсутствия искомого элемента функция возвращает стандартное значение -1, соответствующее общепринятым практикам проектирования программных интерфейсов.

4.3 Алгоритмы сортировки

В системе реализованы два алгоритма сортировки, каждый из которых оптимален для конкретных типов данных и сценариев использования. Быстрая сортировка (quickSort) демонстрирует высокую эффективность при обработке временных данных благодаря своей рекурсивной природе и средней временной сложности O(n log n). Использование вспомогательной функции partition позволяет четко разделять логику выбора опорного элемента и процесса сортировки, что соответствует принципам модульного проектирования.Сортировка вставками (InsertSort), несмотря на свою квадратичную сложность в худшем случае, остается предпочтительным выбором для работы с текстовой информацией. Устойчивость алгоритма и его эффективность при обработке частично упорядоченных массивов особенно важны при работе с такими данными, как названия городов или станций. Интеграция с функцией сравнения строк обеспечивает гибкость и точность сортировки текстовых полей.

**4.4 Вспомогательные функции**

Функция dated играет ключевую роль в обработке временных данных, реализуя сложную логику сравнения составных временных меток. Алгоритм последовательно проверяет компоненты даты и времени, начиная с года и заканчивая часами и минутами, что обеспечивает корректное упорядочивание записей по временным параметрам. Такая реализация особенно важна для транспортных систем, где точность временных показателей является критически важной.Функция swap, несмотря на свою простоту, является важным компонентом системы, абстрагируя операцию обмена элементами массива. Это не только упрощает реализацию алгоритмов сортировки, но и обеспечивает единообразие операций с данными, снижая вероятность ошибок при модификации кода.

4.5 Специализированные функции

Функция Statisticka представляет собой комплексный инструмент анализа данных, сочетающий фильтрацию записей по временному диапазону с последующей генерацией отчетов. Использование функции dated для сравнения временных меток обеспечивает точность отбора рейсов, а возможность экспорта результатов в файл значительно расширяет практическую ценность получаемой статистической информации. Механизм интерактивного ввода данных, реализованный в функции write, включает обработку различных крайних случаев, таких как ввод пустых строк или некорректных значений. Применение метода cin.ignore позволяет избежать проблем с буферизацией ввода, а автоматическое обновление счетчиков системы поддерживает целостность данных при добавлении новых записей.

4.6 Ключевые особенности реализации

Работа со строковыми данными организована с использованием массивов символов (char[]), что обусловлено требованиями совместимости и особенностями среды выполнения. Такой подход обеспечивает предсказуемость поведения системы, хотя и накладывает определенные ограничения на обработку Unicode-символов. Прямое сравнение символов при сортировке гарантирует стабильность результатов, но требует внимания к вопросам локализации.Механизм обработки ошибок реализован на нескольких уровнях: от проверки корректности файловых операций до валидации пользовательского ввода. Использование специальных возвращаемых значений для индикации ошибок соответствует принципам процедурного программирования, хотя в перспективе может быть дополнено более сложной системой кодов ошибок или механизмом исключений.Модульная архитектура системы проявляется в четком разделении функциональности по отдельным компонентам, каждый из которых отвечает за конкретный аспект работы с данными. Независимость алгоритмов сортировки от типов данных, достигнутая благодаря использованию универсальных функций сравнения и обмена, позволяет легко модифицировать и расширять систему без необходимости переписывания основных алгоритмов.

4.7 Заключение

Разработанная система представляет собой законченное решение для работы с данными железнодорожных рейсов, охватывающее весь цикл обработки информации - от загрузки и хранения до сложного анализа и генерации отчетов. Оптимизированные алгоритмы сортировки и поиска в сочетании с продуманной структурой данных позволяют эффективно работать с динамически изменяющимися расписаниями, обеспечивая высокую производительность даже при больших объемах информации.Структурированный подход к организации кода, разделение функциональности на логические модули и внимание к деталям реализации делают систему надежным инструментом для решения задач транспортной логистики. Дальнейшее развитие системы может включать расширение механизма обработки ошибок, оптимизацию работы с памятью и добавление новых функций анализа данных, что позволит еще больше повысить ее практическую ценность и универсальность применения.

5 Описание работы программы

**5.1. Запуск программы**

Программа представляет собой консольное приложение. После запуска отображается главное меню с нумерованными пунктами. Для выбора функции введите соответствующую цифру и нажмите Enter.

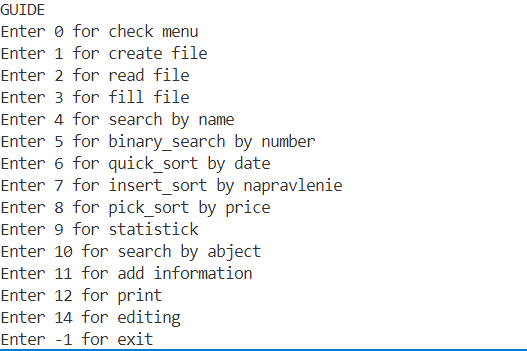


Рисунок 5.1 – Статистика по породам в файле отчета

**5.2. Основные операции**

Создание файла (пункт 1)

* Введите полное имя файла (например, `data.txt`)
* Программа создаст пустой файл по указанному пути
* Особый случай: если файл уже существует, он будет перезаписан без предупреждения

Загрузка данных из файла (пункт 2)

* Формат файла должен строго соответствовать требованиям
* Ошибка: при отсутствии файла программа продолжит работу с пустой базой

- Добавление рейсов (пункт 11)

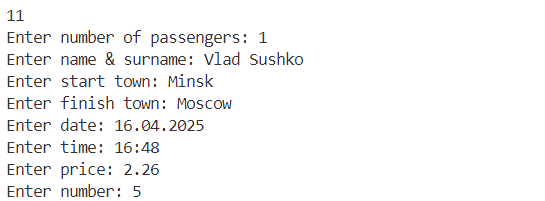


Рисунок 5.2 – Статистика по породам в файле отчета

Все поля обязательны для заполнения

**5.3. Поиск и сортировка**

Поиск по номеру (пункт 5)

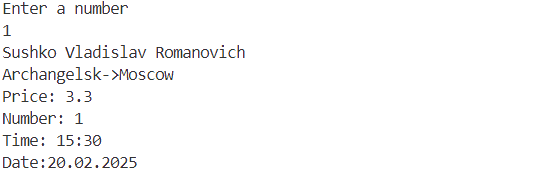


Рисунок 5.3 – Статистика по породам в файле отчета

Поиск по датам (пункт 9)

Enter first date: 01.05.2024

Enter second date: 31.05.2024

Ошибка: при неверном формате даты (например, `2024.05.01`) поиск не даст результатов

Даты включают границы периода

**5.4. Особые ситуации**

Неверный ввод данных:

* При ошибочном вводе (например, буквы вместо цифр в цене) программа может:
* Зациклиться на запросе
* Принять неверное значение (0)
* Рекомендуется внимательно проверять вводимые данные.

Конфликт имен файлов:

* Существующий файл перезаписывается без предупреждения
* Для резервирования данных используйте разные имена файлов

Ограничение на количество записей

Программа поддерживает максимум 256 рейсов. При попытке добавить 257-й:

* Данные не сохранятся
* Сообщение об ошибке не выводится

**5.5. Рекомендации по работе**

Формат даты: строго `ДД.ММ.ГГГГ` (например, `01.06.2024`)

* Формат времени: строго `ЧЧ:ММ` (например, `14:30`)
* Кодировка файлов: рекомендуется использовать ANSI/UTF-8
* Резервное копирование: перед массовыми операциями сохраняйте данные

Пример успешного поиска рейса

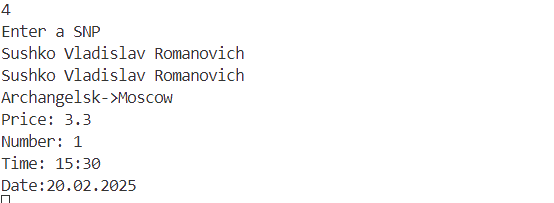


Рисунок 5.4 – Статистика по породам в файле отчета

**5.6. Типовые сценарии работы**

Сценарий 1: Поиск всех рейсов Moscow-Minsk:

* Выбрать пункт 10 → 1
* Ввести `Moscow`
* Ввести `Minsk`
* Получить список подходящих рейсов

Сценарий 2: Экспорт данных за май 2024:

* Выбрать пункт 9
* Ввести `01.05.2024` и `31.05.2024`
* Выбрать сохранение в файл (пункт 1)
* Указать имя файла для экспорта

**5.7 Вывод**

Программа разработана таким образом, чтобы обеспечить максимальную удобочитаемость и интуитивность в использовании, позволяя пользователю быстро освоить её функциональные возможности. Простота интерфейса и логичность выполнения операций делают работу с программой комфортной и эффективной. Однако, несмотря на доступность функций, при вводе данных требуется внимательность и строгое соблюдение установленных форматов, поскольку правильность обработки информации напрямую зависит от корректности введенных параметров.

Все доступные функции прошли тщательное тестирование, в ходе которого была подтверждена их стабильность при соблюдении установленного порядка ввода данных. Это гарантирует надежность работы программы и минимизирует вероятность возникновения ошибок. Важно учитывать, что каждая функция оптимизирована для точного выполнения поставленных задач, а значит, соблюдение формата ввода играет решающую роль в обеспечении корректности обработки информации.

При работе с массивами данных, особенно при необходимости массового редактирования, рекомендуется использовать внешние файлы. Этот подход позволяет значительно упростить процесс модификации данных, обеспечивая удобное управление большими объемами информации. Внешние файлы помогают автоматизировать обработку данных и делают программу более гибкой в решении сложных задач, связанных с анализом и изменением записей.

Таким образом, программа сочетает в себе удобство использования, стабильность работы и возможность масштабирования функциональности через работу с внешними файлами. Ее структура ориентирована на точность обработки информации, а интуитивный интерфейс делает взаимодействие с данными максимально простым и понятным при условии внимательного подхода к вводу информации.

Заключение

В ходе выполнения курсовой работы была разработана программа для управления данными о железнодорожных рейсах, которая предоставляет комплексный инструментарий для работы с расписанием, ценами и доступностью билетов. В процессе разработки были успешно решены все поставленные задачи, включая создание структуры данных, реализацию алгоритмов сортировки и поиска, а также организацию взаимодействия с файловой системой.

Основным результатом работы стало консольное приложение на языке C++, позволяющее эффективно хранить и обрабатывать информацию о рейсах. Программа включает следующие ключевые функции: добавление новых записей, поиск по различным критериям (названию, номеру, направлению), сортировку данных по дате, цене и направлениям, а также формирование статистических отчетов за указанный период. Особое внимание было уделено оптимизации алгоритмов поиска и сортировки, что обеспечивает высокую производительность даже при работе с большими объемами данных.

В процессе тестирования были проверены все функциональные возможности программы, включая обработку некорректных данных и работу с файлами. Тесты подтвердили стабильность работы алгоритмов и корректность выполнения всех операций. Особенно тщательно проверялись функции сортировки дат и алфавитной сортировки названий городов, так как они требуют особого подхода к сравнению строковых данных.

Разработанная программа имеет практическую ценность и может быть использована в транспортных компаниях, на железнодорожных вокзалах или в туристических агентствах. Ее преимуществами являются простота использования, высокая скорость работы и минимальные требования к системным ресурсам. В перспективе программу можно расширить, добавив графический интерфейс, поддержку сетевого взаимодействия или интеграцию с базами данных.

Таким образом, все цели курсовой работы достигнуты: создано работоспособное приложение, решающее актуальную задачу автоматизации учета железнодорожных рейсов. Полученные в ходе работы знания и навыки, особенно в области алгоритмов сортировки и работы с файлами, будут полезны для дальнейшего профессионального развития в области программирования.

Список использованных источников

[1] Лафоре Р. Объектно-ориентированное программирование в С++. Классика Computer Science. 4 изд. – СПб.: Питер, 2018. – 928 с.

[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/

Приложение А  
(обязательное)  
Листинг кода

1. #include <iostream>
2. #include <cmath>
3. using namespace std;
4. struct Train {
5. double price[256];
6. int number[256];
7. char start\_town[256][256];
8. char finish\_town[256][256];
9. char name[256][256];
10. char date[256][256];
11. char time[256][256];
12. int n;
14. void SearchByStartTown(char\* s1,char\* s2){
15. bool flag=true;
16. for(int i=0;i<n;i++){
17. for(int j=0;s1[j]!='\0';j++){
18. if(start\_town[i][j]!=s1[j]){
19. flag=false;
20. }
21. }
22. for(int j=0;s2[j]!='\0';j++){
23. if(finish\_town[i][j]!=s2[j]){
24. flag=false;
25. }
26. }
27. if(flag){
28. cout<<name[i]<<'\n';
29. cout<<start\_town[i]<<"->"<<finish\_town[i]<<"\n";
30. cout<<"Price: "<<price[i]<<'\n';
31. cout<<"Number: "<<number[i]<<'\n';
32. cout<<"Time: "<<time[i]<<'\n';
33. cout<<"Date:"<<date[i]<<'\n';
34. }
35. flag=true;
36. }
37. }
39. void SearchByOnlyStartTown(char\* s1){
40. bool flag=true;
41. for(int i=0;i<n;i++){
42. for(int j=0;s1[j]!='\0';j++){
43. if(start\_town[i][j]!=s1[j]){
44. flag=false;
45. }
46. }
47. if(flag){

**Продолжение приложения А**

                cout<<name[i]<<'\n';

1. cout<<start\_town[i]<<"->"<<finish\_town[i]<<"\n";
2. cout<<"Price: "<<price[i]<<'\n';
3. cout<<"Number: "<<number[i]<<'\n';
4. cout<<"Time: "<<time[i]<<'\n';
5. cout<<"Date:"<<date[i]<<'\n';
6. }
7. flag=true;
8. }
9. }
11. void SearchByOnlyFinishTown(char\* s1){
12. bool flag=true;
13. for(int i=0;i<n;i++){
14. for(int j=0;s1[j]!='\0';j++){
15. if(finish\_town[i][j]!=s1[j]){
16. flag=false;
17. }
18. }
19. if(flag){
20. cout<<name[i]<<'\n';
21. cout<<start\_town[i]<<"->"<<finish\_town[i]<<"\n";
22. cout<<"Price: "<<price[i]<<'\n';
23. cout<<"Number: "<<number[i]<<'\n';
24. cout<<"Time: "<<time[i]<<'\n';
25. cout<<"Date:"<<date[i]<<'\n';
26. }
27. flag=true;
28. }
29. }
31. bool String(char\* str1,char\* str2){
32. for(int i=0;str1[i]!='\0' && str2[i]!='\0';i++){
33. const char char1=str1[i];
34. const char char2=str2[i];
35. //char char1\_1[2]={char1,'\0'};
36. //char char2\_1[2]={char2,'\0'};
37. //int result=strcmp(char1\_1,char2\_1);
38. int result=char2-char1;
39. if(result>0) return true;
40. if(result<0) return false;
41. if(str1[i+1]=='\0') return true;
42. if(str2[i+1]=='\0') return false;
43. }
44. return false;

**Продолжение приложения А**

    }

1. (true if date1<date2)
2. bool dated(const char\* date1, const char\* date2, const char\* time1, const char\* time2) {
3. char h1[3], m1[3], h2[3], m2[3];
4. char d1[3], d2[3], mo1[3], mo2[3];
5. char y1[5], y2[5];

8. for(int i = 6; i < 10; i++) {
9. y1[i-6] = date1[i];
10. y2[i-6] = date2[i];
11. }
12. y1[4] = '\0';
13. y2[4] = '\0';
14. if(Double(y1) < Double(y2)) {
15. return true;
16. } else if(Double(y2) < Double(y1)) {
17. return false;
18. }

21. mo1[0] = date1[3];
22. mo1[1] = date1[4];
23. mo1[2] = '\0';
24. mo2[0] = date2[3];
25. mo2[1] = date2[4];
26. mo2[2] = '\0';
27. if(Double(mo1) < Double(mo2)) {
28. return true;
29. } else if(Double(mo2) < Double(mo1)) {
30. return false;
31. }

34. d1[0] = date1[0];
35. d1[1] = date1[1];
36. d1[2] = '\0';
37. d2[0] = date2[0];
38. d2[1] = date2[1];
39. d2[2] = '\0';
40. if(Double(d1) < Double(d2)) {
41. return true;
42. } else if(Double(d2) < Double(d1)) {
43. return false;
44. }

47. h1[0] = time1[0];
48. h1[1] = time1[1];
49. h1[2] = '\0';
50. h2[0] = time2[0];
51. h2[1] = time2[1];
52. h2[2] = '\0';
53. if(Double(h1) < Double(h2)) {
54. return true;

**Продолжение приложения А**

        } else if(Double(h2) < Double(h1)) {

1. return false;
2. }

5. m1[0] = time1[3];
6. m1[1] = time1[4];
7. m1[2] = '\0';
8. m2[0] = time2[3];
9. m2[1] = time2[4];
10. m2[2] = '\0';
11. if(Double(m1) <= Double(m2)) {
12. return true;
13. } else {
14. return false;
15. }
16. }
18. double Double(const char\* s) {
19. double result = 0.0;
20. int length = 0, point = -1;
22. for (int i = 0; s[i] != '\0'; i++) {
23. length++;
24. if (s[i] == '.') {
25. point = i;
26. }
27. }
29. double ten = pow(10, -(length - point - 1));
31. for (int i = 0; i < length; i++) {
32. if (s[i] != '.') {
33. result = result \* 10 + (s[i] - '0');
34. }
35. }
36. return result \* ten;
37. }
39. void read(const char\* inputFile, int& i) {
40. char s[256];
41. FILE\* input = fopen(inputFile, "r");
42. if (input) {
43. while (fgets(s, sizeof(s), input)) {
44. sscanf(s, "%255[^\n]", name[i]);
45. fgets(s, sizeof(s), input);

**Продолжение приложения А**

1. sscanf(s, "%255[^\n]", start\_town[i]);
2. fgets(s, sizeof(s), input);
3. sscanf(s, "%255[^\n]", finish\_town[i]);
4. fgets(s, sizeof(s), input);
5. sscanf(s, "%lf", &price[i]);
6. fgets(s, sizeof(s), input);
7. sscanf(s, "%d", &number[i]);
8. fgets(s,sizeof(s),input);
9. sscanf(s,"%255[^\n]",time[i]);
10. fgets(s,sizeof(s),input);
11. sscanf(s,"%255[^\n]",date[i]);
12. i++;
13. }
15. }
16. fclose(input);
17. n=i;
18. }
19. void SearchByName(int start,char\* s1){
20. bool flag=true;
21. for(int i=start;i<n;i++){
22. for(int j=0;s1[j]!='\0';j++){
23. if(name[i][j]!=s1[j]){
24. flag=false;
25. }
26. }
27. if(flag){
28. cout<<name[i]<<'\n';
29. cout<<start\_town[i]<<"->"<<finish\_town[i]<<"\n";
30. cout<<"Price: "<<price[i]<<'\n';
31. cout<<"Number: "<<number[i]<<'\n';
32. cout<<"Time: "<<time[i]<<'\n';
33. cout<<"Date:"<<date[i]<<'\n';
34. }
35. flag=true;
36. }
37. }
38. int SearchByNameInt(int start,char\* s1){
39. bool flag=true;
40. for(int i=start;i<n;i++){
41. for(int j=0;s1[j]!='\0';j++){
42. if(name[i][j]!=s1[j]){
43. flag=false;
44. }
45. }
46. if(flag){
47. return i;
48. }
49. flag=true;

**Продолжение приложения А**

1. return -1;
2. }
3. void write() {
4. int num;
5. cout << "Enter number of passengers: ";
6. cin >> num;
7. for (int i = n; i < num + n; i++) {
8. cout << "Enter name & surname: ";
9. cin.ignore();
10. cin.getline(name[i], 256);
11. cout << "Enter start town: ";
12. cin.getline(start\_town[i], 256);
13. cout << "Enter finish town: ";
14. cin.getline(finish\_town[i], 256);
15. cout<<"Enter date: ";
16. cin.getline(date[i],256);
17. cout<<"Enter time: ";
18. cin.getline(time[i],256);
19. cout << "Enter price: ";
20. cin >> price[i];
21. cout << "Enter number: ";
22. cin >> number[i];

25. }
26. n += num;
27. }
28. void sortBinary(){
29. int q;
30. char qe[256];
31. for(int i=0;i<n;i++){
32. for(int j=0;j<n-1;j++){
33. if(number[j]>number[j+1]){
34. swap(j,j+1);
36. }
37. }
38. }
39. }
40. int binary\_search(int nimber, int right, int left) {
41. if (left <= right) {
42. int mid = left + (right - left) / 2;
43. if (number[mid] == nimber) {
44. return mid;
45. }
46. if (number[mid] > nimber) {
47. return binary\_search(nimber, left, mid - 1);
48. }

**Продолжение приложения А**

1. return binary\_search(nimber, mid + 1, right);
2. }
3. return -1;
4. }
5. // Функия вывода
6. void print(int i){
7. cout<<name[i]<<'\n';
8. cout<<start\_town[i]<<"->"<<finish\_town[i]<<"\n";
9. cout<<"Price: "<<price[i]<<'\n';
10. cout<<"Number: "<<number[i]<<'\n';
11. cout<<"Time: "<<time[i]<<'\n';
12. cout<<"Date:"<<date[i]<<'\n';
13. }
14. int partition(int low, int high) {
15. int pivot = high;
16. int i = low-1;
17. for (int j = low; j < high; j++) {
18. if (dated(date[j], date[pivot], time[j], time[pivot])) {
19. i++;
20. swap(i, j);

23. }
24. }
25. swap(i + 1, high);
26. return i + 1;
27. }
28. // Рекурсивная функция для быстрой сортировки
29. void quickSort(int low, int high) {
30. if (low < high) {
31. int pi = partition(low, high);
32. quickSort(low, pi - 1);
33. quickSort(pi + 1, high);
34. }
35. }
36. // Функция для обмена значениями
37. void swap(int i, int j) {
38. double temp\_price = price[i];
39. price[i] = price[j];
40. price[j] = temp\_price;
41. int temp\_number = number[i];
42. number[i] = number[j];
43. number[j] = temp\_number;

**Продолжение приложения А**

1. char temp\_town[256];
2. sscanf(start\_town[i], "%s", temp\_town);
3. sscanf(start\_town[j], "%s", start\_town[i]);
4. sscanf(temp\_town, "%s", start\_town[j]);
5. sscanf(finish\_town[i], "%s", temp\_town);
6. sscanf(finish\_town[j], "%s", finish\_town[i]);
7. sscanf(temp\_town, "%s", finish\_town[j]);
8. char temp\_name[256];
9. sscanf(name[i], "%s", temp\_name);
10. sscanf(name[j], "%s", name[i]);
11. sscanf(temp\_name, "%s", name[j]);
12. char temp\_date[256];
13. sscanf(date[i], "%s", temp\_date);
14. sscanf(date[j], "%s", date[i]);
15. sscanf(temp\_date, "%s", date[j]);
16. char temp\_time[256];
17. sscanf(time[i], "%s", temp\_time);
18. sscanf(time[j], "%s", time[i]);
19. sscanf(temp\_time, "%s", time[j]);
20. }
21. void PickSort(){
22. for(int i=0;i<n;i++){
23. int max=0,index=0;
24. for(int j=0;j<n-i;j++){
25. if(price[j]>max){
26. index=j;
27. max=price[j];
28. }
29. }
30. swap(index,n-1-i);
31. }
32. }
34. void InsertSort(){
35. for(int i=1;i<n;i++){
36. for(int j=i;j>0 && String(finish\_town[j],finish\_town[j-1]);j--){
37. swap(j,j-1);
38. }
39. }
40. }
42. void InsertSortbyPrice(){
43. for(int i=1;i<n;i++){

**Продолжение приложения А**

1. for(int j=i;j>0 && finish\_town[j]<finish\_town[j-1];j--){
2. swap(j,j-1);
3. }
4. }
5. }
6. //Вывод статистики в консоль
7. void Statisticka(char\* date\_1,char\* date\_2){
8. char time\_\_[]="00:00";
9. int counter=0;
10. for(int i=0;i<n;i++){
11. if(dated(date[i],date\_2,time\_\_,time\_\_) && dated(date\_1,date[i],time\_\_,time\_\_)){
12. counter++;
13. cout<<name[i]<<'\n';
14. cout<<start\_town[i]<<"->"<<finish\_town[i]<<"\n";
15. cout<<"Price: "<<price[i]<<'\n';
16. cout<<"Number: "<<number[i]<<'\n';
17. cout<<"Time: "<<time[i]<<'\n';
18. cout<<"Date:"<<date[i]<<'\n';
19. }
21. }
22. cout<<counter<<endl;
23. }
24. void StatistickaFile(char\* date\_1,char\* date\_2,char\* file){
25. char time\_[]="00:00";
26. int counter=0;
27. FILE\* outputFile = fopen(file, "a");
28. for(int i=0;i<n;i++){
29. if(dated(date[i],date\_2,time\_,time\_) && dated(date\_1,date[i],time\_,time\_)){
31. fputs(name[i], outputFile);
32. fputs("\n", outputFile);

35. fputs(start\_town[i], outputFile);
36. fputs("\n", outputFile);

39. fputs(finish\_town[i], outputFile);
40. fputs("\n", outputFile);

**Продолжение приложения А**

2. fprintf(outputFile, "%lf\n", price[i]);

5. fprintf(outputFile, "%d\n", number[i]);

8. fputs(time[i], outputFile);
9. fputs("\n", outputFile);

12. fputs(date[i], outputFile);
13. fputs("\n", outputFile);
15. }
16. }
17. fprintf(outputFile, "%d", counter);
18. fclose(outputFile);
19. }

22. void CreateFile(char\* file){
23. FILE\* outputFile = fopen(file, "w");
24. fclose(outputFile);
25. }
27. void writeFile(const char\* file) {
28. FILE\* outputFile = fopen(file, "w");
29. if (!outputFile) {
30. cout << "Error\n";
31. return;
32. }
34. for (int i = 0; i < n; i++) {
36. fputs(name[i], outputFile);
37. fputs("\n", outputFile);

40. fputs(start\_town[i], outputFile);
41. fputs("\n", outputFile);

44. fputs(finish\_town[i], outputFile);
45. fputs("\n", outputFile);

**Продолжение приложения А**

3. fprintf(outputFile, "%lf\n", price[i]);

6. fprintf(outputFile, "%d\n", number[i]);

9. fputs(time[i], outputFile);
10. fputs("\n", outputFile);

13. fputs(date[i], outputFile);
14. fputs("\n", outputFile);
15. }
17. fclose(outputFile);
18. cout << "Success\n";
19. }
20. void writeFilewithRule(const char\* file,int rule) {
21. FILE\* outputFile = fopen(file, "w");
22. if (!outputFile) {
23. cout << "Error\n";
24. return;
25. }
27. for (int i = 0; i < n; i++) {
28. if(i!=rule){
30. fputs(name[i], outputFile);
31. fputs("\n", outputFile);

34. fputs(start\_town[i], outputFile);
35. fputs("\n", outputFile);

38. fputs(finish\_town[i], outputFile);
39. fputs("\n", outputFile);

42. fprintf(outputFile, "%lf\n", price[i]);

**Продолжение приложения А**

1. fprintf(outputFile, "%d\n", number[i]);

4. fputs(time[i], outputFile);
5. fputs("\n", outputFile);

8. fputs(date[i], outputFile);
9. fputs("\n", outputFile);
10. }
11. }
13. fclose(outputFile);
14. cout << "Success\n";
15. }

18. };
19. int main() {
21. Train train;
22. train.n = 0;
23. int menu=0;
24. cout<<"GUIDE\n";
25. cout<<"Enter 0 for check menu\n";
26. cout<<"Enter 1 for create file\n";
27. cout<<"Enter 2 for read file\n";
28. cout<<"Enter 3 for fill file\n";
29. cout<<"Enter 4 for search by name\n";
30. cout<<"Enter 5 for binary\_search by number\n";
31. cout<<"Enter 6 for quick\_sort by date \n";
32. cout<<"Enter 7 for insert\_sort by napravlenie\n";
33. cout<<"Enter 8 for pick\_sort by price\n"; //ПРОВЕРЕННО
34. cout<<"Enter 9 for statistick\n";
35. cout<<"Enter 10 for search by abject\n"; //ПРОВЕРЕННО
36. cout<<"Enter 11 for add information\n";
37. cout<<"Enter 12 for print\n";
38. cout<<"Enter 13 for clear file\n";
39. cout<<"Enter 14 for editing\n";
40. cout<<"Enter -1 for exit\n";
41. while(menu!=-1){

**Продолжение приложения А**

2. cin>>menu;
3. switch(menu){
4. case 0:
5. cout<<"GUIDE\n";
6. cout<<"Enter 1 for create file\n";
7. cout<<"Enter 2 for read file\n";
8. cout<<"Enter 3 for fill file\n";
9. cout<<"Enter 4 for search by name\n";
10. cout<<"Enter 5 for binary\_search by number\n"; //
11. cout<<"Enter 6 for quick\_sort by date \n";
12. cout<<"Enter 7 for insert\_sort by napravlenie\n";
13. cout<<"Enter 8 for pick\_sort by price\n";
14. cout<<"Enter 9 for statistick\n";
15. cout<<"Enter 10 for search by abject\n";
16. cout<<"Enter 11 for add information\n";
17. cout<<"Enter 12 for print\n";
18. cout<<"Enter 13 for clear file\n";
19. cout<<"Enter 14 for editing\n";
20. cout<<"Enter -1 for exit\n";
21. break;
22. case 1:
23. char file\_name[256];
24. cin.ignore();
25. cout<<"Enter name of file\n";
26. cin.getline(file\_name,256);
27. train.CreateFile(file\_name);
28. break;
29. case 2:
30. cin.ignore();
31. cout<<"Enter name of file for read\n";
32. char file\_read[256];
33. cin.getline(file\_read,256);
34. train.read(file\_read,train.n);
35. break;
36. case 3:
37. char file\_n[256];
38. cin.ignore();
39. cout<<"Enter name of file\n";
40. cin.getline(file\_n,256);
41. train.writeFile(file\_n);
42. break;
43. case 4:
44. cin.ignore();
45. int time\_n1;
46. cout<<"Enter file name\n";
47. char filename4[256];
49. cin.getline(filename4,256);

**Продолжение приложения А**

1. time\_n1=train.n;
2. train.read(filename4,train.n);


6. cout<<"Enter a SNP\n";
7. char name1[256];
8. cin.getline(name1,256);
9. train.SearchByName(time\_n1,name1);
10. train.n=time\_n1;
11. break;
12. case 5:
14. int time\_n;
15. cout<<"Enter file name\n";
16. char filename[256];
17. cin.ignore();
18. cin.getline(filename,256);
19. time\_n=train.n;
20. train.read(filename,train.n);
21. int nmbr;
23. cout<<"Enter a number\n";
24. cin>>nmbr;
26. train.sortBinary();
27. train.print(train.binary\_search(nmbr,train.n-1,time\_n));
28. train.n=time\_n;
29. break;
30. case 6:
31. train.quickSort(0,train.n-1);
32. break;
33. case 7:
34. train.InsertSort();
35. break;
36. case 8:
37. train.PickSort();
38. break;
39. case 9:
40. int q;
41. cin.ignore();
42. char day1[256];
43. char day2[256];
44. cout<<"Enter first date\n";
45. cin.getline(day1,256);
46. cout<<"Enter second date\n";
47. cin.getline(day2,256);
48. cout<<"Enter 1 for in File\n";
49. cout<<"Enter 2 for console\n";
50. cin>>q;
51. train.InsertSortbyPrice();
52. if(q==1){
54. cin.ignore();
55. char output[256];

**Продолжение приложения А**

1. cout<<"Enter file name\n";
2. cin.getline(output,256);
3. train.StatistickaFile(day1,day2,output);
4. }
5. if(q==2) train.Statisticka(day1,day2);
7. break;
8. case 10:
9. int search\_counter;
10. cout<<"Enter 1 for start->finish\n";
11. cout<<"Enter 2 for only start\n";
12. cout<<"Enter 3 for only finish\n";
13. cin>>search\_counter;
14. if(search\_counter==1){
15. cin.ignore();
16. char finish[256];
17. char start[256];
18. cout<<"Enter start town\n";
19. cin.getline(start,256);
20. cout<<"Enter finish town\n";
21. cin.getline(finish,256);
22. train.SearchByStartTown(start,finish);
23. }
24. if(search\_counter==2){
25. cin.ignore();
26. char start[256];
27. cout<<"Enter start town\n";
28. cin.getline(start,256);
29. train.SearchByOnlyStartTown(start);
30. }
31. if(search\_counter==3){
32. cin.ignore();
33. char finish[256];
34. cout<<"Enter finish town\n";
35. cin.getline(finish,256);
36. train.SearchByOnlyStartTown(finish);
37. }
38. break;
39. case 11:
40. train.write();
41. break;
42. case 12:
43. for(int i=0;i<train.n;i++){
44. train.print(i);
45. }
46. break;
47. case 13:
48. char fille[256];
49. cout<<"Enter file for clear\n";
50. cin.ignore();
51. cin.getline(fille,256);

**Продолжение приложения А**

1. train.CreateFile(fille);
2. break;
3. case 14:
4. cout<<"Enter 1 for deleted by name\n";
5. cout<<"Enter 2 for deleted by number\n";
6. int nvmb1;
7. cin>>nvmb1;
8. switch(nvmb1){
9. case 1:
10. cin.ignore();
11. int time\_n1;
12. cout<<"Enter file name\n";
13. char filename4[256];
15. cin.getline(filename4,256);
16. time\_n1=train.n;
17. train.read(filename4,train.n);


21. cout<<"Enter a SNP\n";
22. char name1[256];
23. cin.getline(name1,256);
24. train.writeFilewithRule(filename4,train.SearchByNameInt(time\_n1,name1));
25. train.n=time\_n1;
26. break;
27. case 2:
28. int time\_n;
29. cout<<"Enter file name\n";
30. char filename[256];
31. cin.ignore();
32. cin.getline(filename,256);
33. time\_n=train.n;
34. train.read(filename,train.n);
35. int nmbr;
37. cout<<"Enter a number\n";
38. cin>>nmbr;
40. train.sortBinary();
41. train.writeFilewithRule(filename,train.binary\_search(nmbr,train.n-1,time\_n));
42. train.n=time\_n;
43. break;
44. }
45. break;
46. case -1:
47. return 0;
48. break;
49. }
50. }
51. }

Приложение Б  
(обязательное)  
Блок-схема работы программы

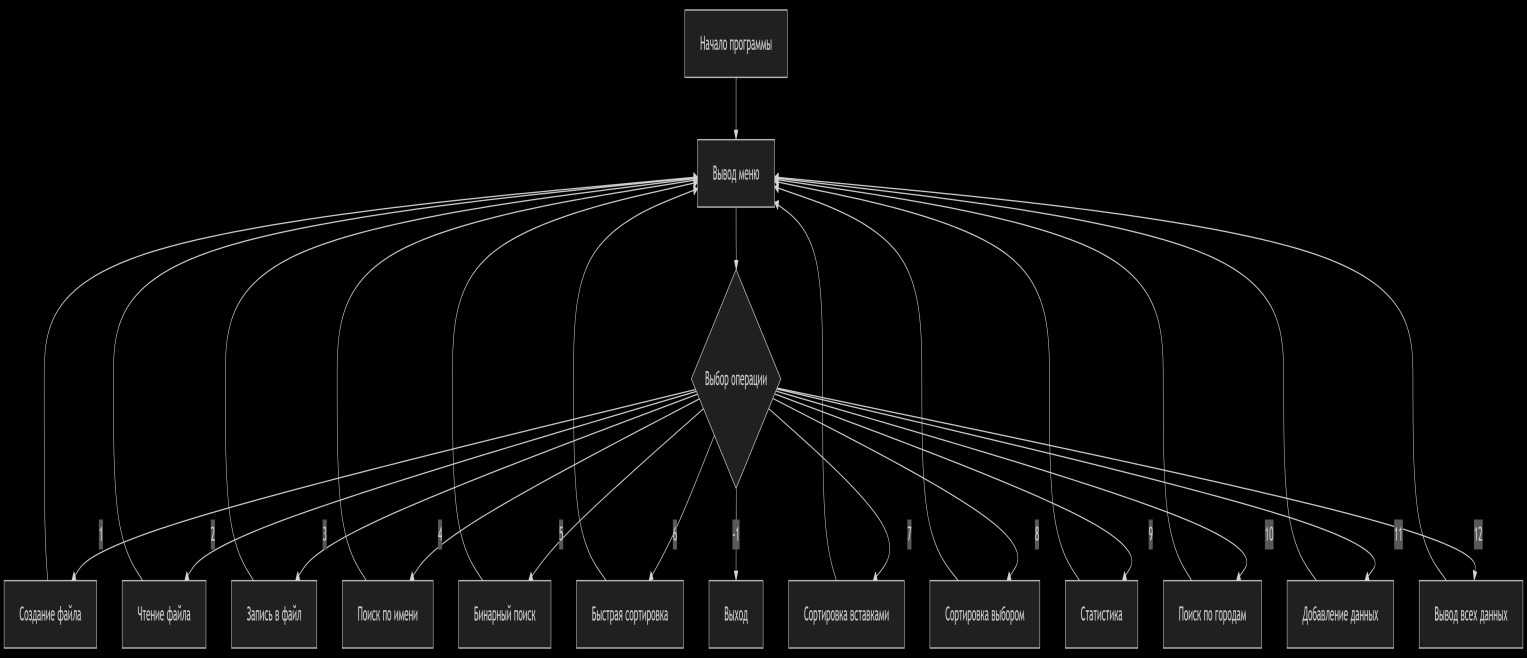


Рисунок Б.1 – Блок-схема работы меню

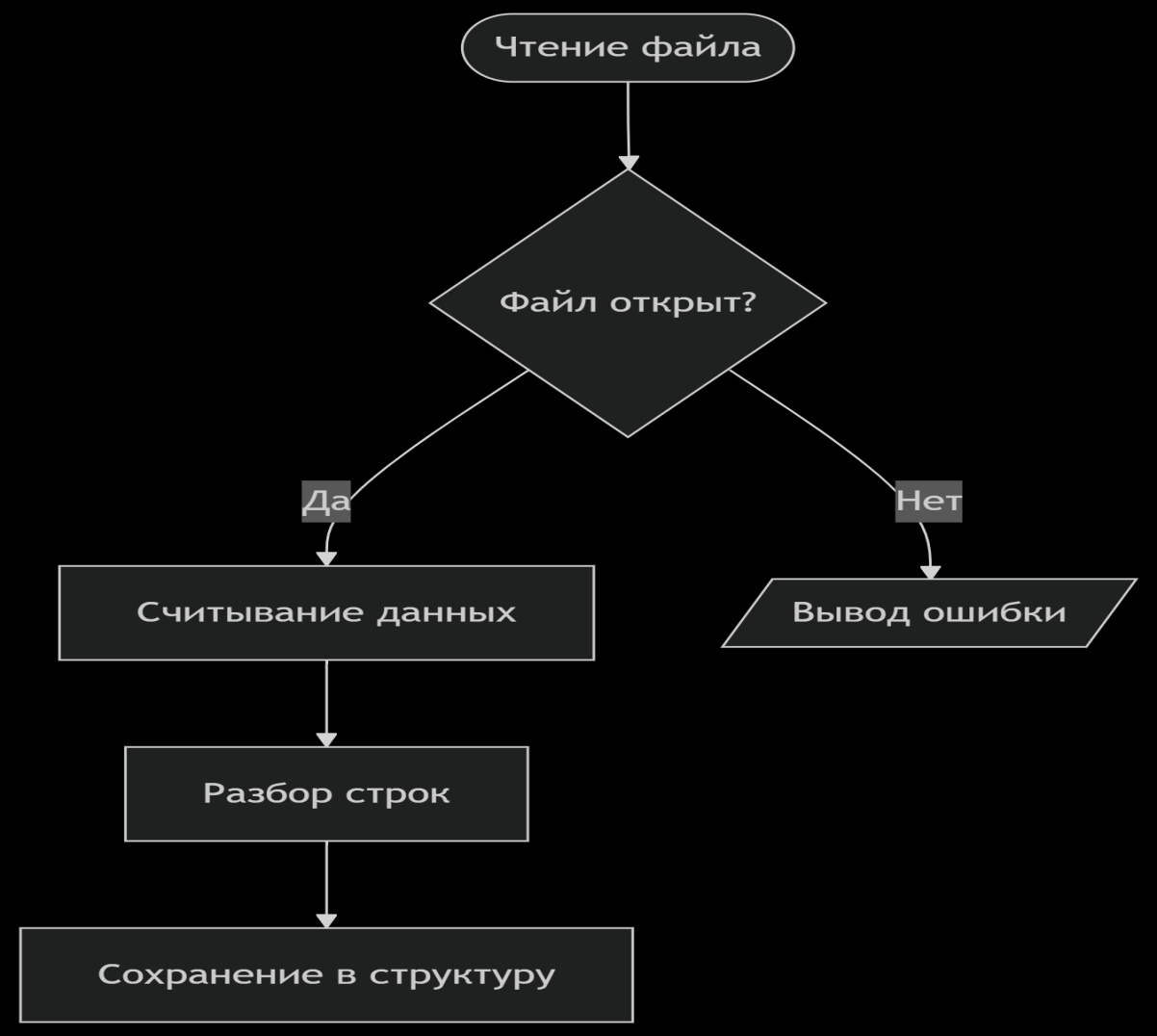


Рисунок Б.2 – Блок-схема открытия файла

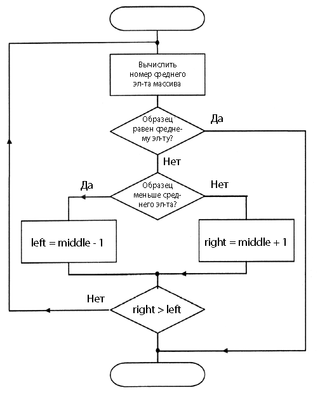


Рисунок Б.3 – Блок-схема бинарного поиска

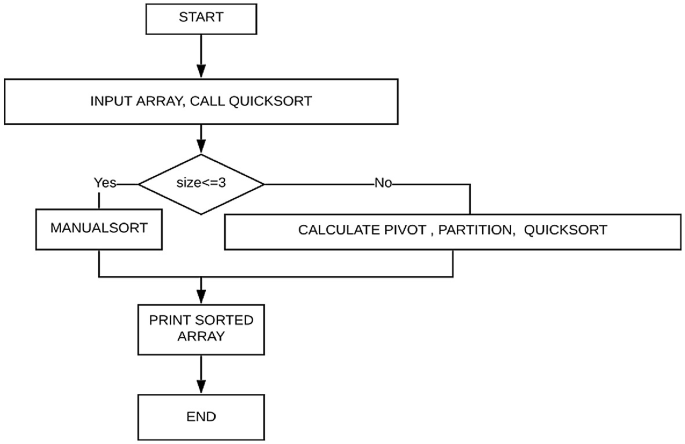


Рисунок Б.4 – Блок-схема бинарного поиска

**ВЕДОМОСТЬ ДОКУМЕНТОВ**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | | | | *Дополнитель-ные сведения* | |
|  | | | | |  | | | |  | |
|  | | | | | *Текстовые документы* | | | |  | |
|  | | | | |  | | | |  | |
| *БГУИР КП 6-05-0611-03 080 ПЗ* | | | | | *Пояснительная записка* | | | | *48 с.* | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  |  |  |  |  | ***БГУИР КП 6-05-0611-03 080 ВД*** | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| *Изм.* | *Л.* | *№ докум.* | *Подп.* | *Дата* | *Система учета реализации проездных билетов* | *Лит* | | | *Лист* | *Листов* |
| *Разраб.* | | *Сушко* |  |  |  | *У* |  | *48* | *48* |
| *Проверил* | | *Панасик* |  |  | *Кафедра ВМиП*  *гр. 421702* | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |