Ульяновский государственный университет

Факультет математики, информационных и авиационных технологий

Кафедра Информационных технологий

Итоговая лабораторная работа по теме:

«Разработка JSON-парсера и системы учета посещаемости»

Выполнил: студент гр. МОАИС-О-25/1
Камалова Е.К.

Ульяновск – 2025 г.

1.1. Цели и задачи

Attendance CLI Tool — это консольное приложение для учета посещаемости студентов, разработанное на C++. Основные задачи:
- Обработка больших объемов данных о посещаемости (до 500,000+ записей)
- Валидация и агрегация данных
- Генерация отчетов и статистики
- Поддержка формата JSON для входных/выходных данных

1.2. Область применения

- Учебные заведения для автоматизации учета посещаемости
- Анализ поведения студентов
- Тестирование производительности алгоритмов обработки данных

2. Технические характеристики

2.1. Стек технологий

- Язык программирования: C++17
- Компилятор: MSVC/GCC/Clang с поддержкой C++17
- Библиотеки: Стандартная библиотека C++ (STL)
- Формат данных*: JSON (собственный парсер)

2.2. Требования к системе

- Операционная система: Windows/Linux/macOS
- Оперативная память: 2+ ГБ (рекомендуется 4+ ГБ для больших файлов)
- Дисковое пространство: 50+ МБ
- Процессор: x64 архитектура

2.3. Архитектура

Attendance CLI Tool
─ main.cpp (Главный модуль, CLI интерфейс)
─ attendance.cpp (Логика работы с посещаемостью)
─ simple_json.cpp (Собственный JSON парсер)
─ utils.cpp (Вспомогательные функции)
─ generator.cpp (Генератор тестовых данных)
─ benchmark_gen.cpp (Генератор данных для бенчмарков)
─ tests_parser.cpp (Тесты JSON парсера)

3. Установка и настройка

3.1. Предварительные требования

1. Установленный компилятор C++17
2. Git для клонирования репозитория

3.2. Сборка проекта

```bash
 Клонирование репозитория
git clone <repository-url> cd
attendance-cli
Или прямая компиляция
g++ -std=c++17 -O2 -I./include main.cpp attendance.cpp simple_json.cpp utils.cpp -o
attendance.exe
```

3.3. Структура каталогов

```
attendance-cli/
├── data/              # Тестовые данные
│   ├── example_valid.json
│   ├── example_huge.json
│   └── generated/
├── include/           # Заголовочные файлы
│   ├── attendance.hpp
│   ├── simple_json.hpp
│   └── utils.hpp
└── src/               # Исходный код
```

4. Использование

4.1. Формат входных данных

Приложение ожидает JSON файл в формате: json

```
[
  {
    "student": "Иванов И.И.",
    "ts": "2025-10-15T08:30:00Z",
    "type": "in"
  },
  {
    "student": "Иванов И.И.",
    "ts": "2025-10-15T14:45:00Z",
    "type": "out"
  }
]
```

Типы событий:

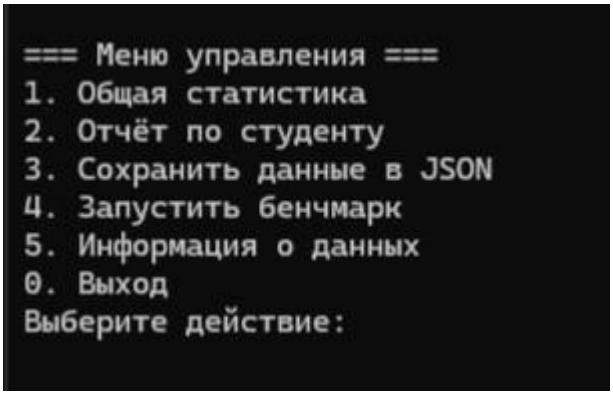
- in - вход
- out - выход
- absence - прогул

4.2. Команды CLI

 # Основные команды
app --help *# Показать справку*
app --input data.json *# Загрузить файл и войти в интерактивный режим* app --input data.json --student "Иванов И.И." *# Отчет по конкретному студенту*
app --input data.json --bench *# Запустить бенчмарк* app --input data.json --validate-only *# Только валидация данных*

 # Генерация тестовых данных
--f or –file *# Загрузить файл*
--n or –count *# Количество файлов JSON*
--e or –error *# Количество ошибок*

Пример:
-f data/example_huge_defects.json -n 500000 -e 500



4.3. Интерактивное меню

После загрузки файла доступно меню:

1. **Общая статистика** - сводка по всем студентам
2. **Отчёт по студенту** - детальная информация о конкретном студенте
3. **Сохранить данные в JSON** - экспорт обработанных данных
4. **Запустить бенчмарк** - тестирование производительности
5. **Информация о данных** - метаинформация 6. **Выход**

5. Описание ключевых алгоритмов:

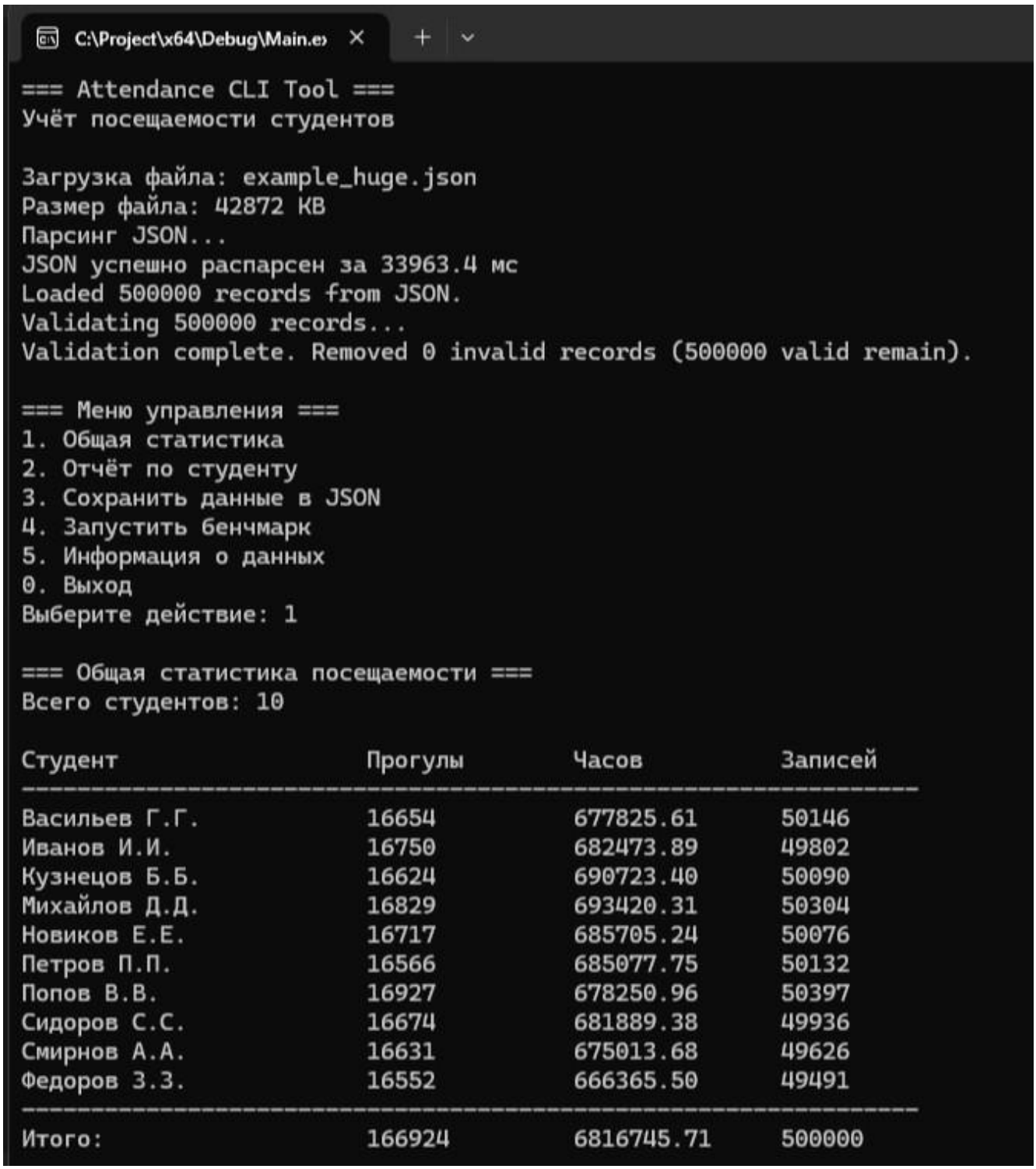
main.cpp - главная функция приложения:

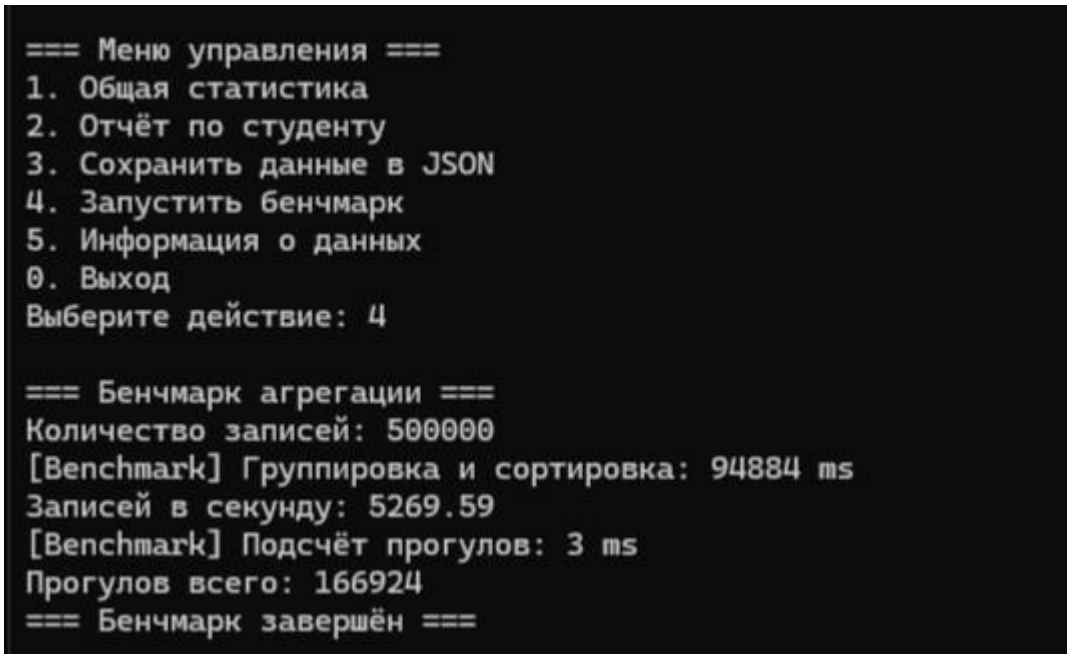
- Реализован CLI интерфейс с аргументами командной строки
- Обработка флагов: --input, --student, --bench, --validate-only
- Интерактивное меню для работы с данными
- Управление потоком выполнения программы **attendance.cpp - ядро логики посещаемости:**
- loadFromJson() - загрузка данных из JSON-структуры
- validateData() - валидация записей с проверкой корректности полей
- benchmarkAggregation() - бенчмаркинг производительности группировки
- printReportByStudent() - генерация отчетов по конкретному студенту

- printGeneralStats() - расчет общей статистики посещаемости **simple_json.cpp - собственный JSON парсер:**
- Recursive descent parser для полного стандарта JSON
- parse() - разбор JSON строки в дерево объектов
- stringify() - сериализация объектов обратно в JSON строку
- Обработка escape-последовательностей и Unicode символов **utils.cpp - вспомогательные утилиты:**
- readFile() / writeFile() - работа с файловой системой
- setupConsoleEncoding() - настройка кодировки консоли (UTF-8)
- getFileSize() - получение размера файла
- u8_adjust() - корректировка ширины для Unicode строк **generator.cpp - генератор тестовых данных:**
- Создание реалистичных данных посещаемости
- Контролируемая генерация ошибок (errorRate)
- Поддержка нескольких форматов файлов
- Генерация stress-тестов (500,000+ записей) **benchmark_gen.cpp - генерация данных для бенчмарков:**
- Создание файла example_huge.json (500,000 записей)
- Равномерное распределение по студентам и типам событий
- ISO 8601 формат временных меток **tests_parser.cpp - тесты JSON парсера:**
- Unit-тесты всех типов JSON данных
- Тесты обработки ошибок
- Бенчмарки производительности парсинга
- Roundtrip тесты (парсинг → сериализация → парсинг)

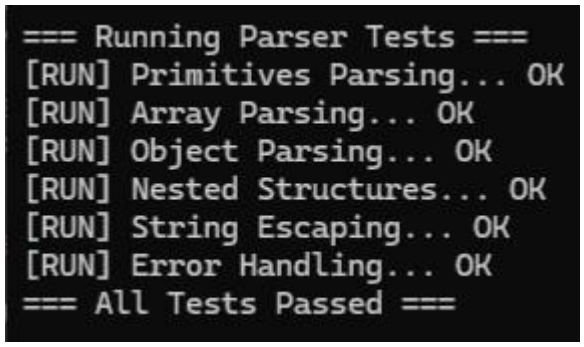
6. Результаты тестов

Обработка 500,000 записей





Tests



Вывод:

Программа работает корректно: обрабатывает до 500,000 записей, генерирует отчеты, валидирует данные
Архитектура модульная: JSON-парсер, обработчик данных, утилиты разделены

Есть проблемы с производительностью: Парсинг JSON занимает 81% времени (DOMподход), можно улучшить интерфейс и функционал.