

## Итоговая лабораторная работа

### Вводная часть

Итоговая работа выполняется в формате небольшого **прикладного проекта**. Цель — объединить и продемонстрировать на практике все темы курса: ввод/вывод и арифметика, логические условия, циклы, массивы/std::vector, строки и файловый ввод/вывод с пользовательским JSON.

### Цель работы и ожидаемые результаты

1. **Цель:** спроектировать и реализовать консольное приложение на C++, способное загружать, валидировать, обрабатывать и сохранять данные в формате JSON; предоставить пользователю удобный текстовый интерфейс, выдерживая требования к надёжности и производительности.
2. **Ожидаемые результаты:**
  - работающая программа (CLI) с понятной справкой (--help),
  - пример(ы) входных и выходных файлов в формате JSON,
  - отчёты/выводы в человекочитаемом виде (таблицы/сводки),
  - замеры времени для ключевых операций и анализ «узкого горлышка»,
  - репозиторий GitHub с документацией и планом реализации.

Строго обязательные требования к проекту

### Репозиторий GitHub (публичный):

- структура:
  - /src     # исходный код .cpp
  - /include # заголовки .h/.hpp
  - /tests    # самотесты
  - /data     # примеры JSON
  - /docs     # Implementation Plan, bench-отчёт, доп. Материалы
  - /scripts  # вспомогательные скрипты (опционально)
- README с инструкцией по сборке/запуску и демонстрационным сценарием;
- .gitignore; осмысленные коммиты; один Pull Request (эмуляция code review);

## **Implementation Plan:**

- файл docs/Implementation\_Plan.md (или PDF) с обязательными разделами (см. далее);
- план должен быть достаточно подробным, чтобы по нему можно было реализовать программу;
- допускается форма связного текста (1-2 стр.), с содержанием таблиц (если это необходимо).

## **Данные и JSON:**

- ручная реализация минимального подмножества JSON: объект, массив, строка, число, true/false/null, базовые экранирования (\", \\, \n);
- Примеры JSON представлены в вариантах, вам необходимо разработать стратегию тестирования и запуска ваших программ (дополнительно сгенерировать до 100000 тысяч JSON файлов заданной структуры, в том числе в которых будут заведомо ошибочные/пустые данные)
- валидация входных данных с понятными сообщениями об ошибках (позиция/контекст);

## **Надёжность и устойчивость:**

- проверки доменов/делений/переполнений/выходов за границы, обработка пустых коллекций, ошибок файлового ввода;
- человеко-дружелюбные сообщения (что произошло и что сделать пользователю дальше);
- корректная работа с вещественными числами (сравнение по EPS).

## **Производительность и узкое горлышко:**

- не менее одной операции, демонстрирующей заметные затраты на больших данных (осознанный hotspot);
- бенчмаркинг в /tests с использованием std::chrono, отчёт docs/bench.md (объём → время);
- анализ причин узкого места и предложенная попытка оптимизации (альтернативный алгоритм/структура/снижение копирований).

## **UX консоли:**

- интерактивное меню и/или флаги командной строки (--help, --input, --output, команды);

- выравнивание колонок, заголовки, единицы измерения, примеры использования;
- повтор ввода при ошибках, подтверждения потенциально опасных действий (перезапись файла).

### Что сдаем:

1. Ссылка на GitHub
2. README с инструкциями и примером сценария запуска.
3. docs/Implementation\_Plan.md (или PDF).
4. Набор данных в /data (минимальный и «крупный» для бенчмарков).
5. Отчёт по замерам: docs/bench.md (методика, таблицы, выводы).
6. Само тесты в /tests и краткий лог их запуска (файл/скрин в docs).
7. Пример форматированного вывода программы (скрин/текст) для демонстрационного сценария.

### Требования к отчёту

Отчёт размещается в docs/ и включает:

- титульный лист/шапку (ФИО, группа, тема);
- краткое описание задачи и среды выполнения (компилятор/OS/IDE);
- структура проекта и схема данных (мини-спецификация JSON);
- описание ключевых алгоритмов;
- выдержки из интерфейсов функций и примеры использования;
- описание обработки ошибок и UX взаимодействия;
- результаты тестов (позитив/негатив/границы); результаты бенчмарков (таблицы/график, если есть);
- выводы и предложения по улучшению.

### Обязательная структура Implementation Plan

1. **Краткое описание задачи** (1–3 предложения).
2. **Цель и ожидаемый результат** (что на выходе: отчёт/файл/статистика и т.д.).
3. **Этапы и задачи реализации** (не менее 6–8 шагов, последовательно: анализ формата → ввод/валидация → парсинг/преобразование → вычисления/алгоритмы → вывод результатов → тестирование/исправления → оформление).
4. **План-график** (сроки по этапам: дата/неделя/занятие).
5. **Ресурсы** (язык/IDE; примеры файлов/спецификации; документация/материалы).

## Критерии оценивания

- **GitHub-дисциплина** (структура, README, PR, релиз) — 10%
- **Implementation Plan** (полнота, реализуемость, связь с кодом) — 15%
- **Структуры данных и JSON** (работоспособность + обоснование) — 20%
- **Надёжность** (валидация, сообщения, граничные случаи) — 15%
- **Качество кода и интерфейсов** (модульность, const&, чистые функции) — 15%
- **Производительность** (бенчмарк, узкое место, попытка оптимизации) — 15%
- **UX консоли** (help, форматирование, удобство) — 10% **Бонус до +5%:** CI сборка, санитайзеры/статический анализ, детальный отчёт профилирования.