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. Самотесты 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.

Варианты заданий.

1) Анализ логов веб-сервера

Ввод/данные. [{ts, ip, method, url, status}], ts — ISO-строка, status — число.

Функционал. Топ-N IP и URL, диапазон дат (мин/макс ts), фильтры по status/method.

Надёжность. Проверка формата ts, диапазона status (100–599), пустых/битых полей.

Производительность. Измерить время частотного анализа на N=10k/50k/100k строк; узкое место — подсчёт топов.

UX/вывод. --top-ip 10, --filter status=404, таблицы с колонками и итогами.

Мини-схема JSON

```
[{
  "ts": "string (ISO 8601)",
  "ip": "string",
  "method": "string (GET|POST|...)",
  "url": "string",
  "status": "number (100..599)"
}]
```

Пример

```
[
  {"ts":"2025-03-14T12:03:21Z","ip":"203.0.113.7","method":"GET","url":"/index.html","status":200},
  {"ts":"2025-03-14T12:03:27Z","ip":"203.0.113.7","method":"GET","url":"/logo.png","status":404}
]
```

2) Кадровые данные (сотрудники)

Ввод/данные. [{fio, birthdate, salary}], birthdate — YYYY-MM-DD, salary — число.

Функционал. Сортировка по полю, вычисление «Возраст» на текущую дату, фильтр по диапазонам.

Надёжность. Валидация дат, зарплат ≥0, корректное сравнение дат.

Производительность. Измерить сортировку и массовый расчёт возрастов; узкое место — сортировка по сложному ключу.

UX/вывод. --sort birthdate --desc, колонка «Возраст», итоги по среднему/медиане зарплат.

Мини-схема JSON

```
[{
  "fio": "string",
  "birthdate": "string (YYYY-MM-DD)",
  "salary": "number >= 0"
}]
```

Пример

```
[
  {"fio":"Иванов И.И.","birthdate":"1999-04-21","salary":85000},
  {"fio":"Петров П.П.","birthdate":"1987-11-03","salary":120000}
]
```

3) Частотный анализ текста

Ввод/данные. {text: "..."} или [{paragraph}]; список стоп-слов в [{stop}].

Функционал. Кол-во слов/предложений/уникальных, топ частот, пересчёт без стоп-слов, распределение по длине.

Надёжность. Нормализация регистра, работа с пунктуацией/пустыми блоками.

Производительность. Измерить частотность; узкое место — разбор большого текста.

UX/вывод. --report freq --top 20, компактные таблицы + сводка.

Мини-схема JSON

```
{
  "text": "string (произвольный большой текст)",
  "stopwords": ["string", "..."]   // опционально
}
```

Пример

```
{
  "text": "В начале было Слово. И Слово было у Бога...",
  "stopwords": ["и","у","в","на","что"]
}
```

4) Шифр Цезаря для текстов

Ввод/данные. [{id, content}], key задаётся параметром; лог операций [{ts, op, key, id}].

Функционал. Шифрование/дешифрование, генерация случайного ключа, пакетная обработка по id.

Надёжность. Ключ в допустимом диапазоне, непустой content, отчёт об ошибках.

Производительность. Измерить посимвольную обработку на больших строках; узкое место — проход по массиву символов.

UX/вывод. --mode enc --key 7 --ids 1,2,3, лог в JSON и итоговая сводка.

Мини-схема JSON

```
[{
  "id": "string",
  "content": "string"
}]
```

Пример

```
[
  {"id":"a1","content":"attack at dawn"},
  {"id":"b2","content":"hello world"}
]
```

5) Конвертация дат

Ввод/данные. [{raw_date}] (разные форматы); целевой формат задаётся флагом.

Функционал. Нормализация в ISO и обратно в локальный, отчёт «валидные/ошибочные».

Надёжность. Разбор неоднозначностей, високосные годы, пустые/битые даты.

Производительность. Измерить массовую конверсию; узкое место — много ветвлений парсинга.

UX/вывод. --to ru_long, две таблицы: конвертировано / ошибки с причиной.

Мини-схема JSON

```
[{
  "raw_date": "string (разные форматы, в т.ч. текстовые)"
}]
```

Пример

```
[
  {"raw_date":"2025-11-30"},
  {"raw_date":"30 ноября 2025"},
  {"raw_date":"11/30/2025"}
]
```

6) Поиск и замена в текстовых объектах

Ввод/данные. [{name, content}].

Функционал. Поиск подстроки, замена, отчёт по каждому name, опция «только поиск».

Надёжность. Пустые строки, регистр, отсутствие совпадений, «сухой прогон».

Производительность. Измерить на контенте большой длины; узкое место — множественные проходы.

UX/вывод. --find foo --replace bar --dry-run, таблица: файл, найдено, заменено.

Мини-схема JSON

```
[{
  "name": "string (идентификатор файла)",
  "content": "string (содержимое)"
}]
```

Пример

```
[
  {"name":"readme.txt","content":"Hello foo"},
  {"name":"data.csv","content":"id,name\n1,foo\n2,bar"}
]
```

7) Трекер задач (todo)

Ввод/данные. [{id, title, due, priority, group, done}].

Функционал. Добавить/изменить/удалить, сортировки по priority/due, фильтр по group, отчёт о просроченных.

Надёжность. Уникальные id, валидные даты, priority ∈ {low, mid, high}.

Производительность. Измерить массовые фильтры+сортировки; узкое место — много сравнений дат.

UX/вывод. --report overdue, форматированные списки с счётчиками.

Мини-схема JSON

```
[{
  "id": "string",
  "title": "string",
  "due": "string (YYYY-MM-DD or ISO)",
  "priority": "string (low|mid|high)",
  "group": "string",
  "done": "boolean"
}]
```

Пример

```
[
  {"id":"t1","title":"Сделать отчёт","due":"2025-12-01","priority":"high","group":"work","done":
]
```

```
:false}
```

8) Мини-система тестирования (quiz)

Ввод/данные. [{id, text, options[], correct[], difficulty, time_limit}].

Функционал. Выбор сложности, случайная выборка, подсчёт баллов, таймер (логика без реального sleep).

Надёжность. Проверка непротиворечивости options/correct, пустых полей.

Производительность. Измерить проверку ответов на большом пуле; узкое место — выборка и проверка множественных ответов.

UX/вывод. --level hard --n 20, сводка баллов и разбор ошибок.

Мини-схема JSON

```
[{
  "id": "string",
  "text": "string",
  "options": ["string", "..."],
  "correct": [0],
  "difficulty": "string (easy|medium|hard)",
  "time_limit": "number (сек.)"
}]
```

Пример

```
[
  {"id":"q1","text":"2+2=?","options":["3","4","5"],"correct":[1],"difficulty":"easy","time_limit":
]
```

```
:30}
```

9) Учёт посещаемости

Ввод/данные. [{student, ts, type}], type ∈ {in, out, absence}.

Функционал. Сводка по студентам/группам, опоздания, процент посещаемости.

Надёжность. Валидация ts, согласованность пар in/out, пустые поля.

Производительность. Измерить группировки по дате/студенту; узкое место — агрегации.

UX/вывод. --student "Иванов" --period week, таблицы с процентами.

Мини-схема JSON

```
[{
  "student": "string",
  "ts": "string (ISO 8601)",
  "type": "string (in|out|absence)"
}]
```

Пример

```
[
  {"student":"Иванов","ts":"2025-10-01T08:59:10Z","type":"in"},
  {"student":"Иванов","ts":"2025-10-01T10:01:05Z","type":"out"}
]
```

10) Каталог медиа

Ввод/данные. [{id, title, author, year, tags[], rating}].

Функционал. Поиск по подстроке/тегам, топ-N по rating, поиск дублей.

Надёжность. Диапазоны year, rating ∈ [0;10], пустые title.

Производительность. Измерить поиск по подстроке; узкое место — множественные сравнения строк.

UX/вывод. --find "дюна" --top 5, аккуратные колонки.

Мини-схема JSON

```
[{
  "id": "string",
  "title": "string",
  "author": "string",   // или director
  "year": "number",
  "tags": ["string", "..."],
  "rating": "number (0..10)"
}]
```

Пример

```
[
  {"id":"bk1","title":"Дюна","author":"Ф. Герберт","year":1965,"tags":["sci-fi"],"rating":9.2}
]
```

11) Складские остатки

Ввод/данные. [{sku, ts, movement, qty}], movement ∈ {in, out}.

Функционал. Текущий остаток по sku, отрицательные остатки, топ расходуемых.

Надёжность. Положительный qty, существование sku, сортировка по ts.

Производительность. Измерить пересчёт по длинной истории; узкое место — суммирование по множеству SKU.

UX/вывод. --stock --sku ABC-123, итоги и предупреждения.

Мини-схема JSON

```
[{
  "sku": "string",
  "ts": "string (ISO 8601)",
  "movement": "string (in|out)",
  "qty": "number (>0)"
}]
```

Пример

```
[
  {"sku":"BOLT-M6","ts":"2025-09-01T10:00:00Z","movement":"in","qty":500},
  {"sku":"BOLT-M6","ts":"2025-09-05T12:00:00Z","movement":"out","qty":120}
]
```

12) Аналитика продаж

Ввод/данные. [{id, ts, items:[{sku, qty, price}]}].

Функционал. Выручка по дню/неделе, средний чек, топ товаров.

Надёжность. qty>0, price≥0, не пустой items.

Производительность. Измерить агрегирование большого числа позиций; узкое место — многократные суммирования.

UX/вывод. --report weekly --top 5, сводные таблицы.

Мини-схема JSON

```
[{
  "id": "string",
  "ts": "string (ISO 8601)",
  "items": [{"sku":"string","qty":"number (>0)","price":"number (>=0)"}]
}]
```

Пример

```
[
  {"id":"r1001","ts":"2025-10-10T12:03:00Z","items":[{"sku":"A1","qty":2,"price":199.9},
]
```

```
,{"sku":"B2","qty":1,"price":50}]}
```

13) Геометрический классификатор точек

Ввод/данные. {polygon:[{x,y}], points:[{x,y}]}.

Функционал. Для каждой точки: IN/ON/OUT (с EPS), площадь/периметр полигона.

Надёжность. Дегенеративные случаи (коллинеарность/дубли), корректные EPS.

Производительность. Измерить классификацию большого набора точек; узкое место — проход по рёбрам для каждой точки.

UX/вывод. --classify --top-on 10, табличный вывод.

Мини-схема JSON

```
{
  "polygon": [{"x":"number","y":"number"}],
  "points":  [{"x":"number","y":"number"}],
  "eps": "number (например, 1e-9)"
}
```

Пример

```
 {
   "polygon":[{"x":0,"y":0},{"x":5,"y":0},{"x":5,"y":3},{"x":0,"y":3}],
   "points":[{"x":2,"y":1},{"x":5,"y":1},{"x":7,"y":1}],
   "eps":1e-9
 }
```

14) Телеметрия датчиков

Ввод/данные. [{sensor, ts, value}].

Функционал. Мин/макс/среднее, медиана, аномалии по скользящему окну.

Надёжность. Пропуски/дубликаты ts, валидность value.

Производительность. Измерить окно и медиану; узкое место — пересчёт оконных метрик.

UX/вывод. --sensor X --window 60s, сводка + список аномалий.

Мини-схема JSON

```
[{
  "sensor": "string",
  "ts": "string (ISO 8601)",
  "value": "number"
}]
```

Пример

```
[
  {"sensor":"s1","ts":"2025-06-01T00:00:00Z","value":12.3},
  {"sensor":"s1","ts":"2025-06-01T00:01:00Z","value":12.9}
]
```

15) Анализ URL

Ввод/данные. [{url}], разбор на {scheme, domain, path, params}.

Функционал. Частотность доменов, поиск дубликатов URL, фильтр по параметрам.

Надёжность. Валидность схемы/домена, экранирование спецсимволов.

Производительность. Измерить разбор множества длинных URL; узкое место — парсинг и сравнения.

UX/вывод. --top-domains 10, отчёт по аномалиям.

Мини-схема JSON

```
[{
  "url": "string"
}]
```

Пример

```
[
  {"url":"https://example.com/search?q=test&lang=ru"},
  {"url":"http://sub.domain.org/path/page.html"}
]
```

16) Телефонный справочник

Ввод/данные. [{name, phone, email, tags[]}].

Функционал. Поиск по префиксу, сортировка по name, дедупликация похожих имён.

Надёжность. Простая проверка форматов phone/email, пустые name.

Производительность. Измерить префиксный поиск; узкое место — сравнения строк.

UX/вывод. --find "Ива", список + число совпадений.

Мини-схема JSON

```
[{
  "name": "string",
  "phone": "string",
  "email": "string",
  "tags": ["string","..."]
}]
```

Пример

```
[
  {"name":"Иванова Анна","phone":"+7-900-111-22-33","email":"anna@example.org",
]
```

```
"tags":["work","sales"]}
```

17) Расписание и конфликты

Ввод/данные. [{course, room, teacher, day, start, end}].

Функционал. Выявление пересечений по аудитории/преподавателю, загрузка по дням.

Надёжность. Корректность интервалов (start<end), диапазоны дней/времени.

Производительность. Измерить поиск конфликтов; узкое место — попарные сравнения ($O(n^2)$).

UX/вывод. --conflicts --by teacher, таблица конфликтов.

Мини-схема JSON

```
[{
  "course": "string",
  "room": "string",
  "teacher": "string",
  "day": "string (Mon..Sun или Пн..Вс)",
  "start": "string (HH:MM)",
  "end": "string (HH:MM)"
}]
```

Пример

```
[
  {"course":"Алгебра","room":"A-101","teacher":"Сидоров","day":"Mon","start":"10:00","end":"11:30"
  {"course":"Физика","room":"A-101","teacher":"Иванова","day":"Mon","start":"11:00","end":"12:30"}
]
```

18) Аналитика поездок

Ввод/данные. [{trip_id, ts_start, ts_end, distance, tariff}].

Функционал. Стоимость, длительность, средняя скорость, выделение выбросов.

Надёжность. ts_end>ts_start, distance≥0, tariff≥0.

Производительность. Измерить расчёт метрик по большому набору; узкое место — множественные вычисления и сортировки.

UX/вывод. --report daily --outliers, сводки и топ выбросов.

Мини-схема JSON

```
[{
  "trip_id": "string",
  "ts_start": "string (ISO 8601)",
  "ts_end": "string (ISO 8601)",
  "distance": "number (км >=0)",
  "tariff": "number (за км >=0)"
}]
```

Пример

```
[
  {"trip_id":"T-1","ts_start":"2025-10-10T08:00:00Z","ts_end":"2025-10-10T08:35:00Z",
]
```

```
,"distance":12.4,"tariff":18.0}
```


19) Котировки (OHLC)

Ввод/данные. [{ts, open, high, low, close, volume}] в порядке времени.

Функционал. Скользящие средние (2 окна), точки пересечения, максимальная просадка.

Надёжность. Монотонность ts, отношения low≤{open,close}≤high.

Производительность. Измерить расчёт индикаторов; узкое место — многократные оконные проходы.

UX/вывод. --sma 5 --sma 20 --crosses, таблица сигналов.

Мини-схема JSON

```
[{
  "ts": "string (ISO 8601 или epoch ms)",
  "open": "number",
  "high": "number",
  "low": "number",
  "close": "number",
  "volume": "number"
}]
```

Пример


```
[
  {"ts":"2025-09-01T10:00:00Z","open":101.2,"high":102.0,"low":100.8,"close":101.7,"volume":15320}
]
```


20) Лимиты запросов к API

Ввод/данные. [{ts, ip, endpoint}].

Функционал. Подсчёт запросов в окне, выявление нарушителей лимита, топ IP.

Надёжность. Корректность ts, устойчивость к неравномерным интервалам.

Производительность. Измерить скользящее окно на большом потоке; узкое место — поддержка оконных счётчиков.

UX/вывод. --window 60 --limit 100, список нарушений и частотный отчёт.

Мини-схема JSON

```
[{
  "ts": "string (ISO 8601)",
  "ip": "string",
  "endpoint": "string"
}]
```

Пример

```
[
  {"ts":"2025-10-05T12:00:01Z","ip":"198.51.100.9","endpoint":"/v1/items"},
  {"ts":"2025-10-05T12:00:02Z","ip":"198.51.100.9","endpoint":"/v1/items"}
]
```