Отчёт по программе, разработанной на C++ в парадигме ООП.

Структура проекта

```
project/
       |
       +-bin/             # собранный бинарный файл
       +-container/       # файлы, описывающие структуру container
       +-objects/         # файлы, описывающие все объекты(структуры, описывающие
числа)
       +-tests/           # файлы с тестовыми входными данными.
       +-utils/           # заголовочный файл с общедоступными статическими функциями
       +-load-testing.sh  # шелл-скрипт для тестирования времени работы проекта.
       +-main.cpp         # точка входа в приложение
```

Спецификация

Спецификация ВС

- **Operating System**: Arch Linux
- **Kernel**: Linux 5.14.7-arch1-1
- **Architecture**: x86-64
- **RAM**: 16Gb

Спецификация средств разработки

- **IDE**: Clion(v2021.2.2)
- **Библиотеки**:
    - stdio.h
    - math.h
    - unistd.h
    - string.h
    - time.h
- **Средство сборки**: CMake(v3.21.3)

Дополнительный флаг `--random-input`

Была реализована функция сохранения сгенерированных входных данных в файл для дальнейшей отладки, для того, чтобы воспользоваться данной функцией необходимо указать флаг `--random-input` и предоставить название файла.

Пример: `./project -r 100 --random-input generated_input.txt -o output.txt` - при данном вводе контейнер заполнится 100 случайно сгенерированными объектами и этот ввод запишится в файл `generated_input.txt`, а вывод программы - в файл `output.txt`. Это позволяет быстро генерировать входные тесты и сразу записывать и входные данные и выходные в нужные файлы.

Характеристики проекта

- Количество заголовочных файлов: 7
- Количество программых объектов: 6

- Размер исходных файлов: ~ 20 Kb
- Размер исполняемого файла: ~ 117 Kb
- Время выполнения программы для различных входных данных

Флаг (-r)	Время выполнения(sec)
10	0.000218
100	0.001552
1000	0.013051
10000	0.524102

Расчет времени выполнения программы

Для расчета времени работы берется среднее арифметическое от 10 запусков программы. Шел скрипт для проверки расположен в корне проекта в файле `load-testing.sh`

Сравнение с предыдущей программой

Различия во времени работы программы для 1 и 2 проектов

Флаг (-r)	Время выполнения(sec) для 1 проекта	Время выполнения(sec) для 2 проекта
10	0.000208	0.000218
100	0.001341	0.001552
1000	0.011584	0.013051
10000	0.537891	0.524102

Программа с классами работает в среднем немного медленнее, это может быть связано с тем, что при использовании классов и наследования создаются виртуальные таблицы(vtables). Для вызова переопределенной функции необходимо пройти по виртуальным таблицам, что требует дополнительного времени. Виртуальные функции - это мощное средство, но цена этому — производительность. Количество заголовочных файлов осталось таким же, так как код на C в функциональной парадигме и на C++ с использованием ООП использует заголовочные файлы одинаковым образом. Количество файлов `.cpp` уменьшилось на 1 по причине рефакторинга кода, т.к. теперь точка является чистой структурой и не имеет в себе методов, а координаты считываются внутри метода `readNumber`.

С точки зрения написания больших проектов, ООП может быть удобнее, чем функциональная парадигма, так как ООП позволяет описывать отношения между объектами и рассуждать о задаче в контексте объектов. Однако, есть примеры больших проектов, которые написаны в функциональной парадигме, например ядро Linux. Поэтому, при выборе парадигмы программирования для своего проекта нужно отталкиваться от своих предпочтений и конкретных задач.

В текущей реализации класс `number` является абстрактным и не реализует никаких общих методов, но если бы класс `number` содержал общие параметры для всех чисел, то `number` имел бы виртульные методы, которые переопределялись бы в наследниках.

Структура текущего архитектурного решения