Отчёт по программе, разработанной на NASM.

Структура проекта

```
project/
       |
       +-bin/             # собранный бинарный файл
       +-asm/             # файлы на языке nasm
       +-c_files/         # файлы, на языке C
       +-tests/           # файлы с тестовыми входными данными
       +-docs/            # документация
       +-c2nasm.sh        # скрипт для конвертации кода на c в код на nasm
       +-objconv          # собранный бинарный файл программы objconv
```

Спецификация

Спецификация ВС

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

Спецификация средств разработки

- **IDE**: VIM
- **Библиотеки**:
  - stdio.h
  - math.h
  - unistd.h
  - string.h
  - time.h
- **Средство сборки**: CMake(v3.21.3), GCC(v11.1.0), NASM(v2.15.05)

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 на языке C: 7
- Количество программых объектов на языке C: 1
- Размер исходных файлов: ~ 20 Kb
- Размер исполняемого файла: ~ 25 Kb
- Время выполнения программы для различных входных данных

Флаг (-r)	Время выполнения(sec)
10	0.000571
100	0.001887
1000	0.037060
10000	0.718191

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 с предыдущими проектами

Флаг (-r)	Время выполнения(sec) для 1 проекта	Время выполнения(sec) для 2 проекта	Время выполнения(sec) для 3 проекта	Время выполнения(sec) для 4 проекта
10	0.000208	0.000218	0.000300	0.000571
100	0.001341	0.001552	0.002100	0.001887
1000	0.011584	0.013051	0.141000	0.037060
10000	0.537891	0.524102	0.940000	0.718191

Т.к. у меня не получилось написать код на nasm, я решил перекомпилировать код из языка C в язык NASM. Для этого я составил shell-скрипт, с помощью которого код на языке C сначала компилируется в object file, а потом дизассемблируется в код на языке nasm. Основной сложностью стал синтаксис nasm - т.к. objconv дизассемблирует код не в обычный nasm формат, то пришлось вручную вырезать все лишние компоненты с помощью утилиты sed. Так же мне пришлось сделать код на C еще более соответствующим C, иначе gcc не мог его компилировать.

Т.к. при компиляции я не пользовался флагами оптимизации, то код на nasm не работает быстрее, чем код на C/C++. Однако, когда я тестировал флаги оптимизации код получался намного более быстрым и производительным. С точки зрения написания проектов, ассемблер не очень удобен, т.к. в больших проектах необходима возможность абстракции от базовых функций(таких как прямая работа со стеком, управление кучей и т.д.), что ассемблер не может себе позволить. Основной точкой применения ассемблера в современном мире является ускорение текущих компонентов систем и программ. Например, часть языка Go написана на ассемблере, что позволило очень сильно ускорить язык и сделать его более "легковесным". Другим применением ассемблера являются компьютерные игры и графика - при создании игр, особенно под консоли, архитектура которых заранее известна, есть смысл писать код на ассемблере, для получения максимальной производительности.