

**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук
Образовательная программа бакалавриата «Программная инженерия»

СОГЛАСОВАНО

Преподаватель департамента
программной инженерии ФКН,
кандидат компьютерных наук

 _____ Р.А.Нестеров
_____ 11_мая 2023 г.

УТВЕРЖДАЮ

Академический руководитель
образовательной программы
«Программная инженерия»
профессор департамента программной
инженерии, кандидат технических наук

 _____ В.В. Шилов
_____ 11_мая_2023 г.


**ВСТРАИВАЕМЫЙ ПРОФИЛИРОВЩИК
ПРОГРАММНОГО КОДА НА ЯЗЫКЕ C++**

Пояснительная записка

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.04-01 01-1-ЛУ

Исполнитель
студент группы БПИ214

 _____ / Е.К.Фортов/
_____ 11_мая_2023 г.

Подп. и дата	
Инв. № дубл.	
Взам. Инв. №	
Подп. и дата	
Инв. № подл.	

УТВЕРЖДЕН
RU.17701729.04.04-01 01-1-ЛУ

ВСТРАИВАЕМЫЙ ПРОФИЛИРОВЩИК ПРОГРАММНОГО КОДА НА ЯЗЫКЕ C++

Пояснительная записка
RU.17701729.04.04-01 01-1-ЛУ

Листов 19

<i>Подп. и дата</i>	
<i>Инв. № дубл.</i>	
<i>Взам. инв. №</i>	
<i>Подп. и дата</i>	
<i>Инв. № подл</i>	

Оглавление

1. ВВЕДЕНИЕ.....	3
1.1. Наименование программы.....	3
1.2. Документы, на основании которых ведётся разработка.....	3
2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ.....	4
2.1. Назначение программы.....	4
2.1.1 Функциональное назначение программы.....	4
2.1.2 Эксплуатационное назначение программы.....	4
2.2. Краткая характеристика области применения.....	5
3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ.....	6
3.1. Постановка задачи на разработку программы.....	6
3.2. Описание алгоритма и функционирования программы.....	6
3.2.1. Общий алгоритм работы программы.....	6
3.2.2. Алгоритм замеров времени частей кода.....	7
3.2.3. Детальный алгоритм работы программы.....	7
3.3. Описание и обоснование выбора метода организации входных и выходных данных.....	10
3.3.1. Описание метода организации входных данных.....	10
3.3.2. Обоснование метода организация входных данных.....	10
3.3.3. Описание метода организации выходных данных.....	11
3.3.4. Обоснование метода организации выходных данных.....	11
3.4. Описание и обоснование выбора метода выбора технических и программных средств.....	11
3.4.1. Описание метода выбора технических и программных средств.....	11
3.4.2. Обоснование метода выбора технических и программных средств.....	11
4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ.....	12
4.1. Ориентировочная экономическая эффективность.....	12
4.2. Предполагаемая потребность.....	12
4.3. Экономические преимущества разработки по сравнению с отечественными или зарубежными аналогами.....	12
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ.....	15
ПРИЛОЖЕНИЕ 1.....	16
ПРИЛОЖЕНИЕ 2.....	16

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

1. ВВЕДЕНИЕ

1.1. Наименование программы

Наименование программы – «Встраиваемый Профилировщик Программного Кода на Языке C++ «FAST_PROFILE»» («Embedded Profiler of C++ Program Code «FAST_PROFILE»»).

1.2. Документы, на основании которых ведётся разработка

Основанием для разработки является учебный план подготовки бакалавров по направлению 09.03.04 «Программная инженерия» и утвержденная академическим руководителем тема курсового проекта.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

2.1. Назначение программы

2.1.1 Функциональное назначение программы

Функциональным назначением программы является предоставление программисту возможности быстро и удобно проводить временные замеры частей своей программы, написанной на C++, во время выполнения программы. Кроме того, разрабатываемая программа создает файлы с результатами проведенных измерений для дальнейшего построения графиков в сторонней программе. Эти данные можно использовать для построения статистики работы исследуемой программы.

2.1.2 Эксплуатационное назначение программы

При написании программы по технической документации разработчик должен оценивать время исполнения программного кода на разных входных данных. Для этого программисту иногда удобно использовать готовый профилировщик кода - программу, которая упрощает проведение замеров времени исполнения отдельных частей кода. К сожалению, использование существующих профилировщиков может приводить к затруднениям по причине довольно длительной настройки, а также внедрения большого числа дополнительных инструкций и команд.

Более того, большинство из них предлагают только графический интерфейс, что делает невозможным тестирование программного кода в консоли, например, если программа тестируется на удаленном сервере с доступом только по протоколу ssh. «FAST_PROFILE» решает эти проблемы следующим образом:

- Подключается одной командой через заголовочный файл
- Для начала/окончания замеров времени, а также для других функций, таких как построение графиков проведенных замеров и вывод в консоль/файлы логов, используются макросы.

Встраиваемый профилировщик кода упрощает проведение временных тестов программного кода. В результате экономится время как разработчика, так и тестировщика.

2.2. Краткая характеристика области применения

«Встраиваемый Профилировщик Программного Кода на Языке C++ «FAST_PROFILE»» - программа для профилирования любых программ на C++.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

Высоконагруженные серверы, графика в играх, ML-задачи и другие профильные и непрофильные программы — сферы применения данного ПО.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

3.1. Постановка задачи на разработку программы

Из функционального назначения программы следует, что программа должна решать поставленные задачи, описанные ниже:

- начинать замер времени;
- заканчивать замер времени;
- выводить результаты замеров в консоли;
- выводить результаты замеров в отдельный файл (логировать в .txt, .json, .csv);
- создавать файлы с результатами проведенных измерений (на некоторых различных входных данных, код программы остается таким же) для дальнейшего построения графиков на основе этих данных в форматах .csv, .json;
- считать и выводить базовую аналитику результатов (выделение наибольшего и наименьшего времени исполнения того или иного участка кода);
- подсвеченный синтаксис результатов, чтобы отличить вывод программы от вывода логов профайлера
- указывать точность замеров времени (вывод в секундах, миллисекундах, микросекундах, наносекундах);
- добавлять комментарии к i-ому замеру времени при вызове той или иной функции профилировщика;
- выводить справку по командам;
- дополнять функциональность путем наследования главного класса профайлера;
- подключать весь профилировщик путем добавления одного файла с помощью директивы include;

3.2. Описание алгоритма и функционирования программы

3.2.1. Общий алгоритм работы программы

Работа профайлера основана на макросах, через параметры которых пользователь передает аргументы для исполнения функций программы. «Ядро» профайлера состоит из двух классов — Profiler и Logger. Класс Profiler использует встраивание класса Logger. Класс Profiler необходим для непосредственно замеров времени работы того или иного участка кода. Для замеров времени была использована стандартная библиотека chrono. Класс Logger логирует результаты, полученные в Profiler как в локальное хранилище (т.е. в свои поля), так и во внешнее хранилище — файлы log.txt, log.csv & log.json. Для логирования результатов замеров времени (в частности, для .json файлов) была выбрана

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

библиотека `nlohmann/json`. Логирование в `.txt` и `.csv` имплементировано с помощью стандартной библиотеки работы с файлами — `fstream`. Логирование в поля класса `Logger` происходит при вызове соответствующих методов, вызванных через макросы. Логирование в файлы происходит при завершении работы программы — во время вызова деструктора класса `Logger`.

3.2.2. Алгоритм замеров времени частей кода

Для замера времени была использована стандартная библиотека `chrono`. Для получения точного значения текущего времени был использован метод `std::chrono::high_resolution_clock::now()`. Вычисление промежутка времени между двумя временными точками было имплементировано с помощью метода `std::chrono::duration_cast<std::chrono::nanoseconds>`. Использование наносекунд позволяет получать нам наиболее точные результаты, и эта точность не теряется при сохранении результата в поля класса `Logger`: результаты всех замеров времени хранятся в программе именно в наносекундах. Для получения результатов в других размерностях (секундах, миллисекундах, микросекундах) используется стандартная операция деления на константы (заданные с помощью макросов в отдельном файле).

3.2.3. Детальный алгоритм работы программы

Описание всех действий, которые должен (может) осуществить пользователь при эксплуатации данного ПО и сопоставление этих действий действиям в коде профайлера:

- Для начала работы профайлера пользователю надо скачать и подключить 2 файла: `profiler.hpp` & папку `include`. В `profiler.hpp` лежат макросы, необходимые для пользования данным ПО, а также сам класс `Profiler`. В `include` лежит всё остальное:
 - Папка `nlohmann` — папка с заголовочным файлом `json.hpp`, в котором находится весь код, необходимый для работы с данными в формате `.json`. Данный файл был взят у пользователя `nlohmann` с его официальной страницы на гитхабе. Написанная им легковесная библиотека `nlohmann/json` предоставляет простой и удобный интерфейс для взаимодействия с файлами формата `.json` без необходимости писать данные функции обработки `.json` файлов самому. Данное ПО распространяется под лицензией MIT, что дает возможность нам как разработчикам свободно использовать его код в наших программах (правда, необходимо указать автора, что было сделано чуть выше).

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

- Заголовочный файл `color.hpp`, который содержит единственную функцию `std::string changeColor(std::string& str)`, которая принимает на вход строку, которую нужно вывести, оборачивает ее в нужные символы для цветного вывода на экран (необходимо для того, чтобы пользователь смог отличить вывод его программы от вывода профилировщика).
- Заголовочный файл `constants.hpp`, в котором определены константы, необходимые для изменения размерности проведенных замеров времени (эти константы помогают переводить между собой: наносекунды, микросекунды, миллисекунды, секунды). Конвертация из одного формата в другой происходит при помощи арифметических операций над данными макросами. Значения всех макросов не превышают `int`, поэтому результат проводимых результатов точно поместится в тип `int64_t`. Говорящие названия макросов: `NANOSECS_IN_SEC` (`=1,000,000,000`), `NANOSECS_IN_MILLISEC` (`=1,000,000`), `NANOSECS_IN_MICROSEC` (`=1,000`), `MICROSECS_IN_SEC` (`=1,000,000`), `MILLISECS_IN_SEC` (`=1,000`), `MICROSECS_IN_MILLISEC` (`=1,000`).
- Текстовый файл `help.txt` — файл со всеми командами данного профайлера и их описанием. Данный файл является документацией профайлера, которую следует прочитать перед использованием ПО, чтобы в дальнейшем не тратить время на поиски ошибок, совершенных из-за незнания команд профилировщика и того, за что каждая из команд отвечает. Содержит 24 основные команды, которые полностью описывают весь функционал профайлера. Остальные команды являются аналогами некоторых основных команд, что также прописано в файле `help.txt`.
- Заголовочный файл `logger.hpp` — файл, в котором определен класс `Logger`, функция `std::string getTimeString(time_t timestamp)`, необходимая для вывода даты в удобном читаемом формате, функция `bool is_empty(std::ifstream& pFile)`, определяющая, является ли данный файл пустым, а также структура `Data`, необходимая для вывода названий соответствующих данных в файл `log.txt`. Выделение отдельных классов для хранения подобных промежуточных результатов было сделано для избежания конфликта имен с кодом пользователя, для которого профайлер и делает замеры времени. Безусловно, полностью защититься невозможно, однако использование имен типа `profiler.start_time` значительно рациональнее, нежели просто `start_time`.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

Положить папку include надо в одну директорию с тестируемой программой. После этого необходимо подключить профайлер в тестируемую программу с помощью #include «profiler.hpp».

- Далее для получения справки по командам надо написать в коде (как можно раньше) `HELP` — при работе программы произойдет обращение к файлу `help.txt` и на экран выведется его содержимое желтым цветом (это обеспечивается функцией `changeColor(std::string& str`, которая принимает на вход строку, которую нужно вывести, оборачивает ее специальными символами и возвращает эту обертку, которая потом просто передается в стандартный поток вывода).
- Затем необходимо определить часть кода, которая будет тестироваться на предмет времени исполнения. Перед этим куском кода надо поставить макрос `START/STARTP/START_PRINT_LINE/START_PRINT`). Эти команды зафиксируют данный момент времени и положат это значение в поле `startTime` класса `Profiler`. Время будет взято из времени на локальной машине. Для обеспечения точности времени будет использован тип `std::chrono::time_point<std::chrono::high_resolution_clock>`.
- После исследуемого участка кода необходимо поставить макрос `END`. Эта команда положит текущее время в поле `stopTime` класса `Profiler`, посчитает разницу (`stopTime — startTime`), положит это значение в поле `duration_time` класса `Profiler` и сохранит в логах — полях класса `Logger`. При использовании команд `ENDP/END_PRINT` поведение профайлера будет аналогичным команде `END`, только дополнительно текущий результат будет выведен в консоль (также желтым цветом). Логирование (сохранение замера времени) происходит с помощью метода `log(const int64_t& nanoseconds)` в поле `logs`. Также можно остановить замер времени с комментарием — он будет отображен в логах. Это делается с помощью команд `END_WITH_COMMENT(str)/END_COMMENT(str)`, где `str` — комментарий. Он сохраняется в массиве `std::vector<std::string> comments` — поле класса `Logger`. Для логирования с комментарием используется перегруженная функция `log - void log(const int64_t& nanoseconds, const std::string& comment)`. Если требуется и оставить комментарий к замеру времени, и вывести результат на экран, следует воспользоваться командой `ENDP_WITH_COMMENT(str) /`
`END_PRINT_WITH_COMMENT(str) / ENDP_COMMENT(str) /`

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

END_PRINT_COMMENT(str). Алгоритм работы данных команд аналогичен командам выше (симбиоз действий).

- Для (дополнительной) распечатки замеренного результата можно воспользоваться командой PRINT_TIME / PRINT_TIME_IN_NANOSECS / PRINT_TIME_IN_MICROSECS / PRINT_TIME_IN_SECS / PRINT_TIME_IN_MILISECS. Эти команды возьмут результат последнего замера времени (profiler.logger.comments.back() в классе Logger) и выведут результат в нужной размерности на экран желтым цветом. По умолчанию используются миллисекунды (команда PRINT_TIME). Для вывода результата с комментарием используются функции PRINT_TIME_WITH_COMMENT(str) / PRINT_TIME_IN_NANOSECS_WITH_COMMENT(str) / PRINT_TIME_IN_MICROSECS_WITH_COMMENT(str) / PRINT_TIME_IN_SECS_WITH_COMMENT(str) / COMMENT(str), где str — комментарием (локальный, он будет отображен только в консоли и не будет сохраняться в логах. Для сохранения комментария в логи должна быть использована команда END_WITH_COMMENT(str), описанная выше). Алгоритм работы этих команд аналогичен командам, описанным выше.
- Для получения информации о данном ПО пользователь может воспользоваться макросом INFO. Этот макрос выведет литералы, описывающие что есть данный продукт, зачем он нужен, базовые команды, а также контакты его разработчика.
- Для вывода всех имеющихся в данной сессии логов можно воспользоваться макросом SHOW_ALL_LOGS, который вызовет метод profiler.logger.show_all_statistic, который в свою очередь обратится к вектору logs и выведет все его значения на экран в том порядке, в котором пользователь совершал замеры времени.
- Для очистки логов можно воспользоваться макросом CLEAR_LOGS, который удалит все логи в текущей сессии (логи предыдущих сессий останутся в файлах). Этот макрос вызовет метод clear_logs() класса Logger, который в свою очередь вызовет clear() у вектора logs. Для удаления всех файлов логов есть макрос CLEAR_ALL_FILE_LOGS, который вызывает метод clear_log_files() класса Logger, который в свою очередь вызывает 3 раза функцию std::remove(<name_of_log_file>) (эта функция определена в библиотеке cstdio), которая удаляет файлы log.txt, log.json, log.csv. Для удаления как текущих логов (логов текущей сессии работы

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

программы), так и файлов логов, используется макрос `CLEAR_ALL_LOGS`, который делает то же самое, что и макросы `CLEAR_LOGS` и `CLEAR_ALL_FILE_LOGS`.

- Для изменения размерности результатов замеров используется макрос `CHANGE_DIMENSION(str)`, где `str` — нужная размерность логов: "NANOSECS" (значение по умолчанию), "MICROSECS", "MILISECS", "SECS". Данная команда меняет значение поля `format` в классе `Logger` с помощью метода `changeDimension(int8_t format)`, в который передается либо 0 (значение по умолчанию — наносекунды — «NANOSECS»), 1 ("MICROSECS"), 2 ("MILISECS"), 3 ("SECS"). Замечание: данная команда работает только для внешних логов — в файлах. Причем размерность замеров времени в файлах будет соответствовать аргументу последнего вызова данной команды. В локальных логах (вывод в консоль) для изменения стандартной размерности (миллисекунды) следует пользоваться командами `PRINT_TIME_IN_NANOSECS` / `PRINT_TIME_IN_MICROSECS` / `PRINT_TIME_IN_SECS` / `PRINT_TIME_IN_MILISECS` (описаны выше).

3.3. Описание и обоснование выбора метода организации входных и выходных данных

3.3.1. Описание метода организации входных данных

Входные данные для данного ПО — это макросы в тестируемой .cpp программе. Макросы подробно описаны в пункте выше. Также к некоторым макросам можно добавить строковые комментарии, которые кратко опишут, замер какого функционального участка кода был сделан. Подробнее про комментарии в макросах также написано выше.

3.3.2. Обоснование метода организации входных данных

Приведённая организация входных данных полностью отвечает требованиям технического задания и является одной из наиболее простых к использованию.

3.3.3. Описание метода организации выходных данных

Выходные данные данного ПО — это консольные команды, выведенные через стандартный поток ввода (желтым цветом), а также файлы логов в форматах .txt, .csv и .json, записанные с помощью библиотек `fstream` и `nlohmann/json`. Как уже говорилось

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

выше, желтый цвет текста в консоли позволит отличить вывод пользовательской программы от вывода профайлера.

3.3.4. Обоснование метода организации выходных данных

Приведённая организация выходных данных полностью отвечает требованиям технического задания и является одной из наиболее простых и наглядных.

3.4. Описание и обоснование выбора метода выбора технических и программных средств

3.4.1. Описание выбора метода выбора технических и программных средств

Для бесперебойной работы программного продукта требуется компьютер с:

- установленной версией компилятора gcc - 14, clang — 3.4
- операционной системой со стабильной сборкой, выпущенной не позднее 2015 года
- объемом свободной встроенной памяти не меньше 55 МБ,
- объёмом оперативной памяти не меньше 1 ГБ.

3.4.2. Обоснование выбора метода выбора технических и программных средств

Программа очень легковесна, требует минимум ресурсов: предъявленным требованиям удовлетворяет >95% компьютеров от их общего количества.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

4.1. Ориентировочная экономическая эффективность

В рамках данной работы расчет экономической эффективности не предусмотрен.

4.2. Предполагаемая потребность

Данный профилировщик могут использовать все разработчики/тестировщики с компилятором gcc версии не ниже 14 или компилятором clang версии не ниже 3.4, которым нужно быстро протестировать работу части своей программы на предмет времени выполнения. Данный профайлер предлагает простое, быстрое и легковесное решение данной проблемы, упрощая жизнь разработчикам и тестировщикам. Данное ПО может быть использовано специалистами разного уровня: от новичка (например, для замеры времени исполнения кода олимпиадной задачи на больших данных, чтобы иметь гарантию того, что участник верно выбрал алгоритм и временные характеристики его программы соответствуют ограничениям задачи) до профессионала (например, замерить скорость подключения к базе данных в высоконагруженном сервисе и получения/записи данных в нее, чтобы проверить, соответствует ли его программа требованиям в техническом задании).

4.3. Экономические преимущества разработки по сравнению с отечественными или зарубежными аналогами

На момент создания программы наиболее используемыми аналогами в области профилировщиков являются: Callgrind, Google perftools и EasyProfiler.

Общий недостаток всех этих продуктов — их сложность в использовании, которое требует установки соответствующей программы с графическим интерфейсом, далее ее линковка с тестируемой программой, далее вызов определенных конструкций из командной строки. Даже после завершения работы профайлера сложно правильно понять и трактовать полученные результаты. Более того, невозможно понять, как именно работал данный профайлер.

Есть и другие недостатки данных решений — например, это привязка к конкретной операционной системе. Например, callgrind является частью инструмента valgrind, текущих версий которого нет ни для Mac OS, ни для Windows. Google perftools, в свою очередь, не может корректно отрабатывать на малых программах, так как использует довольно старые библиотеки операций над временем в C++. EasyProfiler делает ставку на

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

GUI, без которого работа сервиса становится непонятна из-за большого количества выводимых данных и большого количества команд.

Также, Callgrind и Google perftools для работы требуют прописывать свои флаги компиляции, отличные от тех, чем пользуется программист при компилировании своей программы. Это затрудняет понимание компилирующих опций, усложняя процесс debug-а. EasyProfiler, в свою очередь, использует только стандартные флаги компиляции, которые лишь задают путь к необходимым для работы профайлера библиотекам, не засоряя тем самым терминал. Это весомое преимущество данного сервиса перед другими, и задача разрабатываемого в данной работе профайлера будет «унаследовать» этот принцип, максимально его оптимизировав (по возможности используя только уже встроенные в стандартную библиотеку языка C++ библиотеки).

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

- 1) EasyProfiler — простой профилировщик кода [Электронный ресурс] / Сергей @yse. Режим доступа: <https://habr.com/ru/post/318142/>, свободный. (дата обращения: 10.02.2023)
- 2) ValGrind – содержит профилировщик CallGrind [Электронный ресурс] / Pointfor Services. Режим доступа: <https://valgrind.org/info/platforms.html>, свободный. (дата обращения: 10.02.2023)
- 3) Особенности профилирования программ на C++ [Электронный ресурс] / @svr_91. Режим доступа: <https://habr.com/ru/post/482040/>, свободный. (дата обращения: 10.02.2023)
- 4) Lightweight profiler library for c++ [Электронный ресурс] / @yse. Режим доступа: https://github.com/yse/easy_profiler?ysclid=ldz43zqbit864077970, свободный. (дата обращения: 10.02.2023)
- 5) Статья про профилирование программ [Электронный ресурс] / Mateen Ulhaq. Режим доступа: <https://stackoverflow.com/questions/375913/how-do-i-profile-c-code-running-on-linux>, свободный. (дата обращения: 10.02.2023)
- 6) Профилировщик кода от Google [Электронный ресурс] / @alk. Режим доступа: <https://github.com/gperftools/gperftools?ysclid=ldz4785f5i699002535>, свободный. (дата обращения: 10.02.2023)
- 7) Статья про профилирование программ [Электронный ресурс] / Wikipedia. Режим доступа: [https://en.wikipedia.org/wiki/Profiling_\(computer_programming\)](https://en.wikipedia.org/wiki/Profiling_(computer_programming)), свободный. (дата обращения: 10.02.2023)
- 8) Статья про тестирование и профилирование программ [Электронный ресурс] / Wikipedia. Режим доступа: https://en.wikibooks.org/wiki/Introduction_to_Software_Engineering/Testing/Profiling, свободный. (дата обращения: 10.02.2023)
- 9) Видео про профилирование программ [Электронный ресурс] / Wikipedia. Режим доступа: <https://yandex.ru/video/preview/1757229695302647836?family=yes>, свободный. (дата обращения: 10.02.2023)
- 10) Флаги компиляции callgrind [Электронный ресурс]; Режим доступа: <https://manpages.org/valgrind> свободный. (дата обращения: 10.02.2023)
- 11) Библиотека nlohmann/json [Электронный ресурс]; Режим доступа: <https://github.com/nlohmann/json> свободный. (дата обращения: 10.05.2023)
- 12) Детальное описание лицензии MIT [Электронный ресурс]; Режим доступа: <https://habr.com/ru/articles/310976/> свободный. (дата обращения: 10.05.2023)
- 13) Описание лицензии MIT [Электронный ресурс]; Режим доступа: https://ru.wikipedia.org/wiki/Лицензия_MIT свободный. (дата обращения: 10.05.2023)
- 14) Описание лицензии MIT [Электронный ресурс]; Режим доступа: <https://tproger.ru/articles/whats-difference-between-licenses/?ysclid=lhiamqaaqt382514907> свободный. (дата обращения: 10.05.2023)

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

ПРИЛОЖЕНИЕ 1

ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ КЛАССОВ

Таблица 1.1 – Описание и функциональное назначение классов/структур в файле profiler.hpp

Класс/файл	Назначение
Profiler (class)	Класс необходим для замеров времени того или иного участка кода пользователя. Содержит поля, нужные для временного хранения полученных результатов. Нужен также для снижения риска избежания конфликтов имен временных переменных профилировщика с переменными в пользовательской программе.

Таблица 1.2 – Описание и функциональное назначение классов/структур в файле logger.hpp

Класс/файл	Назначение
Logger (class)	Класс необходим для логирования полученных результатов как в локальное хранилище (т.е. в свои поля), так и во внешнее хранилище — файлы log.txt, log.csv и log.json.
Data (struct)	Необходима для вывода названий соответствующих данных в файл log.txt.

ПРИЛОЖЕНИЕ 2

ОПИСАНИЕ И ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ ПОЛЕЙ, МЕТОДОВ, СВОЙСТВ КЛАССОВ, А ТАКЖЕ ПЕРЕМЕННЫХ И ФУНКЦИЙ В ФАЙЛАХ

Таблица 2.1.1 – Описание полей и свойств класса Logger

Имя	Модификатор доступа	Тип	Назначение
logs	public	std::vector<int64_t>	В данном поле хранятся все полученные в текущей сессии результаты замеров времени работы пользовательского кода. Хранятся данные в наносекундах в не зависимости от формата логов, выбранным

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

			пользователем — это нужно для сохранения точности измерений. Так, если бы при изменении формата данных менялись бы и logs, то при дальнейшем переводе из более крупных величин в более мелкие, точность изменений была бы некорректной.
comments	public	std::vector<std::string>	В данном поле хранятся комментарии, связанные с результатами замеров времени работы пользовательского кода. Если комментарий не было, то на его место ставится «-».
start_time	public	std::time_t	Время начала работы логгера в текущей сессии. Поле инициализируется в конструкторе класса Logger. Инициализация происходит временем в системе (ОС).
end_time	public	std::time_t	Время конца работы логгера в текущей сессии. Поле инициализируется в деструкторе класса Logger. Инициализация происходит временем в системе (ОС).

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

Таблица 2.1.2 – Описание методов класса Logger

Имя	Модификатор доступа	Тип	Аргументы	Назначение
changeDimension	public	void	int8_t format	Изменение размерности логов. 0 — наносекунды, 1 — микросекунды, 2 — миллисекунды, 3 — секунды. Замечание — обратное изменение формата логов из более крупных единиц в более мелкие не принесет потери точности: в logs все данные о замерах времени всегда хранятся в наносекундах вне зависимости от поля format.
log	public	void	const int64_t& nanoseconds	Записывает переданный замер времени в наносекундах в поле logs и «-» в поле comments.
log	public	void	const int64_t& nanoseconds , const std::string& comment	Записывает переданный замер времени в наносекундах в поле logs и переданный комментарий в поле comments.
getMax	public	int64_t	-	Возвращает самый продолжительный замер времени из имеющихся в поле logs.
getMin	public	int64_t	-	Возвращает самый короткий замер времени из имеющихся в поле logs.
clear_logs	public	void	-	Обнуляет поле logs
clear_log_files	public	void	-	Удаляет файлы log.txt, log.csv,

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

				log.json.
show_all_statistics	public	void	-	Выводит все элементы поля logs в консоль желтым цветом (с помощью функции std::string changeColor(std::string& str)).

Таблица 2.2.1 – Описание полей и свойств класса Profiler

Имя	Модификатор доступа	Тип	Назначение
logger	public	Logger	Сохранение, обработка и вывод полученных измерений.
startTime	public	std::chrono::time_point<std::chrono::high_resolution_clock>	Начало замера времени.
stopTime	public	std::chrono::time_point<std::chrono::high_resolution_clock>	Конец замера времени.
duration_time	public	std::chrono::duration<ll, std::nano>	Продолжительность работы кода (сам замер времени).

Таблица 2.2.2 – Описание переменных файла profiler.hpp

Имя	Модификатор доступа	Тип	Назначение
STR_NO_DUPLICATION	public	std::string	Промежуточное хранилище строковых литералов для передачи текста в функцию std::string changeColor(std::string& str).

Таблица 2.2.3 – Описание функций файла profiler.hpp

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

Имя	Модификатор доступа	Тип	Аргументы	Назначение
getGracefulDoubleString	public	Std::string	double str	Возвращает строку из числа типа double, округляя его до 3х знаков после запятой.

Таблица 2.3.1.1 – Описание переменных файла logger.hpp

Имя	Модификатор доступа	Тип	Назначение
STR_NO_DUPLICATION_Q	public	std::string	Переменная для промежуточного хранения литералов (строк) для передачи в функцию std::string changeColor(std::string& str).

Таблица 2.3.1.2 – Описание функций файла logger.hpp

Имя	Модификатор доступа	Тип	Аргументы	Назначение
is_empty	public	bool	std::ifstream & pFile	Возвращает true, если переданный файл пуст. False в противном случае.
getTimeString	public	std::string	time_t timestamp	Возвращает дату в красивом формате в виде строки.

Таблица 2.3.2 – Описание функций файла color.hpp

Имя	Модификатор доступа	Тип	Аргументы	Назначение
changeColor	public	std::string	std::string& str	Оборачивает строку в специальные символы для того, чтобы она выводилась в консоль в желтом цвете.

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.04-01 01				
Инв. № подл.	Подп. и дата	Взам. Инв. №	Инв. № дубл.	Подп. и дата

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

[illegible]