ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа бакалавриата «Программная инженерия»

СОГЛАСОВАНО
Научный руководитель,
доцент департамента программной
инженерии факультета
ко наук, кандидат
ких наук

__ __Р.А.Нестеров
___ _11_мая_ 2023 г.

УТВЕРЖДАЮ
Академический руководитель
образовательной программы
«Программная инженерия»
про мента программной
инж ат технических наук

___ В.В. Шилов
_ __11_мая__2023 г.

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

ВСТРАИВАЕМЫЙ ПРОФИЛИРОВЩИК ПРОГРАММНОГО КОДА НА ЯЗЫКЕ C++

Программа и методика испытаний

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.04-01 01-1-ЛУ

Исполнитель
студент группы БПИ214
___ ___/ Е.К.Фортов/
_____ __11_мая_2023 г.

Москва 2023

УТВЕРЖДЕН
RU.17701729.04.04-01 01-1-ЛУ

ВСТРАИВАЕМЫЙ ПРОФИЛИРОВЩИК ПРОГРАММНОГО КОДА НА ЯЗЫКЕ C++

Программа и методика испытаний

RU.17701729.04.04-01 01-1-ЛУ

Листов 18

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

Москва 2023

Оглавление

1. ОБЪЕКТ ИСПЫТАНИЙ

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. Краткая характеристика области применения

«Встраиваемый Профилировщик Программного Кода на Языке C++ «FAST_PROFILE»» - программа для профилирования любых программ на C++. Высоконагруженные серверы, графика в играх, ML-задачи и другие профильные и непрофильные программы — сферы применения данного ПО.

2. ЦЕЛЬ ИСПЫТАНИЙ

Целью испытаний является проверка корректности выполнения программой функций, перечисленных в разделе «Требования к программе».

3. ТРЕБОВАНИЯ К ПРОГРАММЕ

3.1. Требования к функциональным характеристикам

3.1.1. Требования к составу выполняемых функций

Программа должна давать пользователю возможность выполнять следующие функции:

- 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;
- удалять логи (как локально, так и глобально)

3.1.2. Требования к организации входных данных

Входные данные — это исходный код, в который встраиваются нужные команды профилировщика (исходный код может быть представлен в файлах с расширениями .cpp & .h). Исходных код должен отрабатывать корректно, без предупреждений компилятора и ошибок.

В свою очередь, команды профилировщика, внедряемые в код тестируемой программы, — это макросы или отдельные функции, которые могут принимать

дополнительную информацию (например, комментарии к тому или иному замеру времени) в виде строк и целочисленных значений.

Касательно требований к входным данным, пользователю необходимо ознакомиться с внутренней справкой профайлера, которая описывает, какие входные данные принимает та или иная функция.

3.1.3. Требования к организации выходных данных

Результат работы профайлера на тестируемом коде должен содержать следующее (цвет вывода — по умолчанию стандартный, если не был указан другой в основном коде) :

1) вывод результатов проведенных замеров (вывод в консоль + .csv & .json файлы для дальнейшей постройки графиков в сторонних сервисах)
2) вывод базовой аналитики (максимальное/минимальное время тестов)
3) вывод во внешний .txt файл отчета о проведенном тестировании (комментарии)
4) вывод справки по командам (при вызове соответствующей команды профилировщика)

Если программа, на которой проводились замеры времени, отработала с некорректным завершением (код возврата не 0), профайлер может ничего не вывести и не сохранить проведенные исследования во внешний файл.

3.2. Требования к интерфейсу

Графический интерфейс у данного сервиса фактически отсутствует, так как все команды прописываются именно в исходном файле.

Справка по командам профайлера содержится в прилагаемом .txt файле, также возможен вызов справки из исходного кода в консоль с помощью соответствующей команды в исходном файле.

3.3. Требования к надежности

Программа не должна аварийно завершаться при любом наборе входных данных (если не подразумевается отладка приложения).

4. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

Состав программной документации:

1. «Встраиваемый Профилировщик Программного Кода на Языке C++». Техническое задание (ГОСТ 19.201-78 [2])
2. «Встраиваемый Профилировщик Программного Кода на Языке C++». Программа и методика испытаний (ГОСТ 19.301-79 [3])
3. «Встраиваемый Профилировщик Программного Кода на Языке C++». Текст программы (ГОСТ 19.401-78 [4])
4. «Встраиваемый Профилировщик Программного Кода на Языке C++». Пояснительная записка (ГОСТ 19.404-79 [5])
5. «Встраиваемый Профилировщик Программного Кода на Языке C++». Руководство оператора (ГОСТ 19.505-79 [6])

5. СРЕДСТВА И ПОРЯДОК ИСПЫТАНИЙ

5.1. Технические средства, используемые во время испытаний

Ноутбук Macbook Pro 14 дюймов, M1.

5.2. Программные средства, используемые во время испытаний

На ноутбуке имеется:

- ОС Mac OS Ventura 13.2.1
- ОЗУ 16 ГБ
- компилятор clang последней стабильной версии

5.3. Порядок проведения испытаний

Испытания должны проводиться в следующем порядке:

- проверка требований к программной документации,
- проверка требований к функциональным характеристикам,
- проверка требований к интерфейсу,
- проверка требований к надёжности.

5.4. Загрузка программы для испытания

Тестируемая программа — test.cpp. В одну директорию с ней положены заголовочный файл profiler.h и директория include со всеми зависимостями.

6. МЕТОДЫ ИСПЫТАНИЙ

6.1. Испытание выполнения требований к программной документации

Состав программной документации проверяется визуально, проверяется наличие программной документации в системе LMS. Также визуально проверяется соответствие документации требованиям ГОСТ.

Все документы удовлетворяют представленным требованиям.

6.2. Испытание выполнения требований к функциональным характеристикам

6.2.1. Начало/конец замера времени, вывод результатов в консоль, в файлы логов вместе с базовой аналитикой полученных результатов

В данном случае для начала замера времени используется макрос STARTP, для окончания — макрос ENDP_WITH_COMMENT(str), где str — комментарий. Файлы логов создаются и заполняются автормаически.

```
#include <iostream>
#include <pthread.h>  // c lib for posix threads
#include <cstdio>
#include "profiler.hpp"  // handwritten simple profiler

#define CYCLES 1'000'000'000

> void* func(void* arg1) {…

> void* func2(void* arg1) {…


int main(void) {
    pthread_t th1, th2;

    STARTP;
    pthread_create(&th1, NULL, &func, NULL);
    pthread_create(&th2, NULL, &func2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);
    ENDP_WITH_COMMENT("hahahjdhhd");
```

```
Started...
1000000000
1000000000
982.771 milisecs
```

log.txt

```
1   ------------------------------------------------
2   Profiler started at: Sat May 13 23:14:50 2023
3   Profiler ended at: Sat May 13 23:14:51 2023
4   ----- LOGS -----
5   DIMENSION: nanosecs
6   967682708 hahahjdhhd
7   ----- STATS -----
8   Min value: 967682708 ; Max value: 967682708
9   ------------------------------------------------
10
11
```

log.json > ...

```
1   [
2       {
3           "comments": "hahahjdhhd, ",
4           "end_time": "2023-05-13 23:14:51",
5           "logs": "967682708, ",
6           "start_time": "2023-05-13 23:14:50"
7       }
8   ]
```

log.csv

```
1   967682708;
2
```

6.2.2. Вывод справки по командам

Используя макрос HELP, выведем справку по командам. Используя макрос INFO, посмотрим информацию о профайлере.

test.cpp > main(void)

```cpp
#include <iostream>
#include <pthread.h>  // c lib for posix threads
#include <cstdio>
#include "profiler.hpp"  // handwritten simple profiler

#define CYCLES 1'000'000'000

void* func(void* arg1) {…

void* func2(void* arg1) {…


int main(void) {
    pthread_t th1, th2;

    STARTP;
    pthread_create(&th1, NULL, &func, NULL);
    pthread_create(&th2, NULL, &func2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);
    ENDP_WITH_COMMENT("hahahjdhhd");
    HELP
    INFO
    // CLEAR_ALL_LOGS;
    return 0;
```

```
egorfortov@macbook-pro code % ./a.out
Started...
1000000000
1000000000
983.387 milisecs
Short overview of all commands:
///////////////// PROFILER HELP /////////////////
Key commands:
    1. START - triggers profiler to start count time
    2. STARTP - START + printing line that profiler started
    3. START_PRINT_LINE,
       START_PRINT = STARTP
    4. END - triggers profiler to finish timer that measured current piece of code
    5. ENDP - the same as previous + print time into console
    6. END_PRINT = ENDP
    7. END_WITH_COMMENT(str) - END + string comment
    8. END_COMMENT(str) = END_WITH_COMMENT(str)
    9. ENDP_WITH_COMMENT(str) - ENDP + string comment
    10. END_PRINT_WITH_COMMENT(str),
        ENDP_COMMENT(str),
        END_PRINT_COMMENT(str) = ENDP_WITH_COMMENT(str)
    11. PRINT_TIME - shows you the result of your measurement in milisecs
    12. PRINT_TIME_IN_MILISECS = PRINT_TIME
    13. PRINT_TIME_IN_NANOSECS,
        PRINT_TIME_IN_MICROSECS,
        PRINT_TIME_IN_SECS - similarly, but with another dimension
    14. PRINT_TIME_WITH_COMMENT(str) - PRINT_TIME + comment in brackets (temporary comment, wil
l NOT be shown in log files)
    15. COMMENT(str) = PRINT_TIME_WITH_COMMENT(str)
    16. PRINT_TIME_IN_NANOSECS_WITH_COMMENT(str),
        PRINT_TIME_IN_MICROSECS_WITH_COMMENT(str),
        PRINT_TIME_IN_SECS_WITH_COMMENT(str) - similarly, but with another dimension
    17. INFO - shows basic info about this profiler
    18. HELP - shows current file
    19. SHOW_ALL_LOGS - shows all previous logs
    20. CLEAR_LOGS - clears all logs in current program
```

```
        PRINT_TIME_IN_SECS - similarly, but with another dimension
    14. PRINT_TIME_WITH_COMMENT(str) - PRINT_TIME + comment in brackets (temporary comment, wil
l NOT be shown in log files)
    15. COMMENT(str) = PRINT_TIME_WITH_COMMENT(str)
    16. PRINT_TIME_IN_NANOSECS_WITH_COMMENT(str),
        PRINT_TIME_IN_MICROSECS_WITH_COMMENT(str),
        PRINT_TIME_IN_SECS_WITH_COMMENT(str) - similarly, but with another dimension
    17. INFO - shows basic info about this profiler
    18. HELP - shows current file
    19. SHOW_ALL_LOGS - shows all previous logs
    20. CLEAR_LOGS - clears all logs in current program
    21. CLEAR_ALL_FILE_LOGS - clears all log files
    22. CLEAR_ALL_LOGS - clear all log files & log in the current program
    23. ANALYZE - prepares files for analyzing (plug, needed for further updated versions of pr
ofiler)
    24. CHANGE_DIMENSION(str) - changes the dimension to:
        NANOSECS (default) - when str = "NANOSECS"
        MICROSECS - when str = "MICROSECS"
        MILISECS - when str = "MILISECS"
        SECS - when str = "SECS"
        Attention! During one runtime the dimension of ALL this runtime's results will be the s
ame
                    (that was set last)

How to use correctly:
    START -> END -> PRINT_TIME -> repeat this cycle;
    All other commands can be written anywhere, but do not forget about logic!

Needed environment:
    C++ version 17+ (i.e. 17, 20, 23, ...)
------- Simple fortov profiler -------
Simple profiler for measuring execution time in .cpp programms;
dev's email: ekfortov@edu.hse.ru%
egorfortov@macbook-pro code %
```

6.2.3. Удаление логов (как локально, так и глобально)

С помощью команды CLEAR_LOGS удалим локальные логи, то есть логи текущей сессии работы программы. При этом ранее полученные логи (в файлах) останутся.

```
test.cpp > main(void)
1   #include <iostream>
2   #include <pthread.h>  // c lib for posix threads
3   #include <cstdio>
4   #include "profiler.hpp"  // handwritten simple profiler
5
6   #define CYCLES 1'000'000'000
7
8   void* func(void* arg1) {…
16
17  void* func2(void* arg1) {…
25
26
27  int main(void) {
28      pthread_t th1, th2;
29
30      STARTP;
31      pthread_create(&th1, NULL, &func, NULL);
32      pthread_create(&th2, NULL, &func2, NULL);
33      pthread_join(th1, NULL);
34      pthread_join(th2, NULL);
35      ENDP_WITH_COMMENT("hahahjdhhd");
36      CLEAR_LOGS;
37      SHOW_ALL_LOGS;
38      return 0;
```

```
egorfortov@macbook-pro code % ./a.out
Started...
1000000000
1000000000
985.205 milisecs
ALL LOGS:
Those are all logs!
egorfortov@macbook-pro code %
```

log.txt

```
1   ------------------------------------------------
2   Profiler started at: Sat May 13 23:14:50 2023
3   Profiler ended at: Sat May 13 23:14:51 2023
4   ----- LOGS -----
5   DIMENSION: nanosecs
6   967682708 hahahjdhhd
7   ----- STATS -----
8   Min value: 967682708 ; Max value: 967682708
9   ------------------------------------------------
10
11  ------------------------------------------------
12  Profiler started at: Sat May 13 23:21:00 2023
13  Profiler ended at: Sat May 13 23:21:01 2023
14  ----- LOGS -----
15  DIMENSION: nanosecs
16  983387084 hahahjdhhd
17  ----- STATS -----
18  Min value: 983387084 ; Max value: 983387084
19  ------------------------------------------------
20
21
```

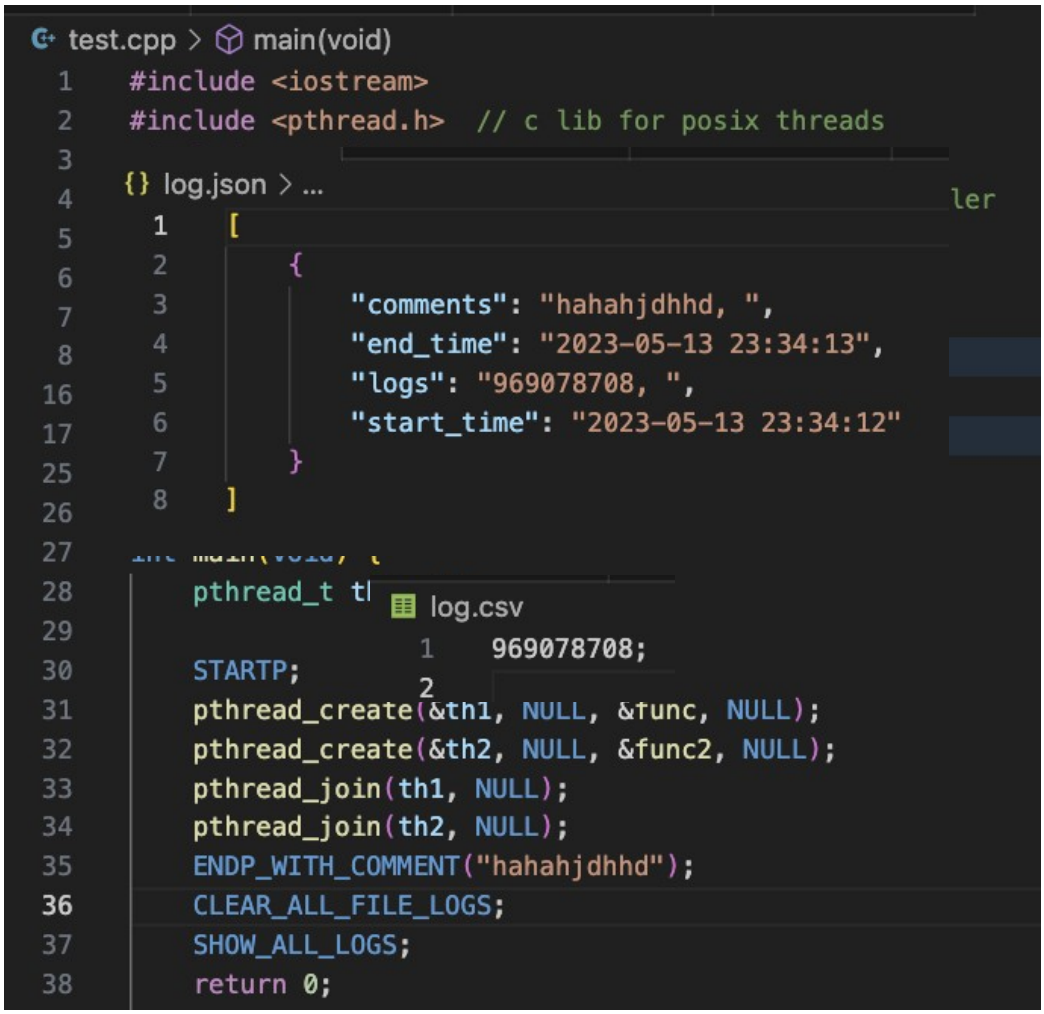
log.csv

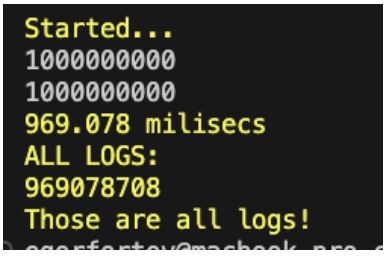
```
1   967682708;
2   983387084;
3
```

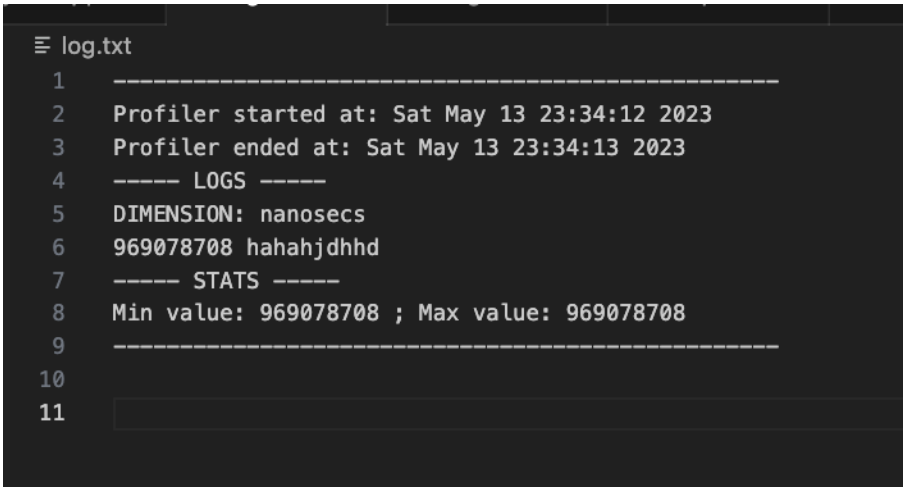
log.json > ...

```
1   [
2       {
3           "comments": "hahahjdhhd, ",
4           "end_time": "2023-05-13 23:14:51",
5           "logs": "967682708, ",
6           "start_time": "2023-05-13 23:14:50"
7       },
8       {
9           "comments": "hahahjdhhd, ",
10          "end_time": "2023-05-13 23:21:01",
11          "logs": "983387084, ",
12          "start_time": "2023-05-13 23:21:00"
13      }
14  ]
```

С помощью команды CLEAR_ALL_FILE_LOGS удалим логи в файлах (фактически, просто удалим предыдущие логи).







При выполнении команды CLEAR_ALL_LOGS очистятся все логи, то есть фактически вызовутся две описанные выше команды: CLEAR_ALL_FILE_LOGS & CLEAR_LOGS.

6.3. Испытание выполнения требований к интерфейсу

Требования к интерфейсу проверяются визуально. Представленный интерфейс приложения в предыдущем разделе удовлетворяет всем требованиям.

6.4. Испытание выполнения требований к надёжности

Программа была протестирована > 100 раз на разных тестовых кодах. Ошибок не обнаружено.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

2) ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

3) ГОСТ 19.301-79 Программа и методика испытаний. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

4) ГОСТ 19.401-78 Текст программы. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

5) ГОСТ 19.404-79 Пояснительная записка. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

6) ГОСТ 19.505-79 Руководство оператора. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Изм.	Номера листов (страниц)				Всего листов (страниц) в документе	№ документа	Входящий № сопроводительного документа и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					