МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Качество и метрология программного обеспечения»

ТЕМА: ИЗМЕРЕНИЕ ХАРАКТЕРИСТИК ДИНАМИЧЕСКОЙ СЛОЖНОСТИ ПРОГРАММ С ПОМОЩЬЮ ПРОФИЛИРОВЩИКА SAMPLER

Студент гр. 7304 _____ Ажель И.В.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

Изучение метода измерения динамической сложности программ с помощью SAMPLER.

Формулировка задания.

1. Ознакомиться с документацией на монитор SAMPLER и выполнить под его управлением тестовые программы test_cyc.c и test_sub.c с анализом параметров повторения циклов, структуры описания циклов, способов профилирования процедур и проверкой их влияния на точность и чувствительность профилирования.

2. Скомпилировать и выполнить под управлением SAMPLER'a программу на C, разработанную в 1-ой лабораторной работе. Выполнить разбиение программы на функциональные участки и снять профили для двух режимов:

   a. измерение только полного времени выполнения программы;
   b. измерение времен выполнения функциональных участков (ФУ).

Убедиться, что сумма времен выполнения ФУ соответствует полному времени выполнения программы.

3. Выявить "узкие места", связанные с ухудшением производительности программы, ввести в программу усовершенствования и получить новые профили. Объяснить смысл введенных модификаций программ.

Примечания.

1. Для трансляции программ следует использовать компиляторы Turbo C++, ver.3.0 (3.1, 3.2).

2. Для выполнения работы лучше использовать более новую версию монитора Sampler_new и выполнять ее на 32-разрядной машине под управлением ОС не выше Widows XP (при отсутствии реальных средств можно использовать виртуальные).

В этом случае результаты профилирования будут близки к реальным.

Если использовать более старую версию монитора Sampler_old, то ее следует запускать под эмулятором DOSBox-0.74. При этом времена, полученные в профилях будут сильно (примерно в 10 раз) завышены из-за накладных затрат эмулятора, но относительные соотношения временных затрат будут корректны.

3. Если автоматически подобрать время коррекции правильно не удается (это видно по большим значениям измерений времени для коротких фрагментов программы, если время коррекции недостаточно, либо по большому количеству нулевых отсчетов, если время коррекции слишком велико), то следует подобрать подходящее время коррекции ручным способом, уменьшая или увеличивая его в нужную сторону.

4. Так как чувствительность SAMPLER'a по времени достаточно высока (на уровне единиц микросекунд), то вводить вспомогательное зацикливание программы обычно не требуется. Но если измеренные времена явно некорректны, следует ввести зацикливание выполнения программы в 10-100 раз. При этом для каждого повторения выполнения программы следует использовать одни и те же исходные данные.

5. Для обеспечения проверки представляемых вами профилей преподавателем необходимо **выполнить нумерацию строк** кода программы, соответствующую нумерации строк, указанной в профиле. У преподавателя нет времени для подсчета номеров строк в отчете каждого студента.

Ход работы.

Работа выполнялась с использованием старого SAMPLER'а, так как использование нового в трех операционных системах (Windows 2000, Windows XP SP1, Windows XP SP3) приводило к ошибке (General Protection Fault in module WIN87EM.DLL), которую не получилось исправить. Так как из не получилось собрать программы с помощью компилятора turbo c++ (видимо проблемы с доступом к библиотеке kernel32.dll), то для выполнения работы использовался компилятор Borland C++ ver 3.1. В качестве операционной системы, на которой производится компиляция, линковка и применение SAMPLER'а использовался виртуальный образ Windows XP Professional SP1 x32.

В тестовые программы подключен файл Sampler.h и расставлены точки измерения. Измененные программы с пронумерованными строками представлены в приложении А.

Результаты профилирования программ TEST_CYC и TEST_SUB представлены на рис. 1 и рис. 2 соответственно.

NN	Имя обработанного файла
1.	TEST_CYC.CPP

Таблица с результатами измерений (используется 13 из 416 записей)

Исх.Поз.	Прием.Поз.	Общее время(мкс)	Кол-во прох.	Среднее время(мкс)
1 : 7	1 : 9	4334.64	1	4334.64
1 : 9	1 : 11	8675.14	1	8675.14
1 : 11	1 : 13	21672.34	1	21672.34
1 : 13	1 : 15	43348.87	1	43348.87
1 : 15	1 : 18	4335.47	1	4335.47
1 : 18	1 : 21	8670.11	1	8670.11
1 : 21	1 : 24	21671.50	1	21671.50
1 : 24	1 : 27	43348.87	1	43348.87
1 : 27	1 : 33	4336.31	1	4336.31
1 : 33	1 : 39	8668.43	1	8668.43
1 : 39	1 : 45	21678.20	1	21678.20
1 : 45	1 : 51	43348.87	1	43348.87

Рисунок 1 - Результаты профилирования файла TEST_CYC

```
------------------------------------------------------------------------
 NN         Имя обработанного файла
------------------------------------------------------------------------
 1.  TEST_SUB.CPP
------------------------------------------------------------------------


 Таблица с результатами измерений ( используется 5 из 416 записей )

------------------------------------------------------------------------
Исх.Поз. Прием.Поз.  Общее время(мкс)  Кол-во прох. Среднее время(мкс)
------------------------------------------------------------------------
1 : 29  1 : 31        433698.18         1       433698.18
------------------------------------------------------------------------
1 : 31  1 : 33        867392.18         1       867392.18
------------------------------------------------------------------------
1 : 33  1 : 35       2168480.87         1      2168480.87
------------------------------------------------------------------------
1 : 35  1 : 37       4336937.43         1      4336937.43
------------------------------------------------------------------------
```

Рисунок 2 - Результаты профилирования файла TEST_SUB

Проанализировав результаты, можно отметить, что на время выполнения циклов влияет только число итераций в них. При этом, зависимость можно считать линейной.

Далее произведем аналогичные действия с программой из лабораторной работы 1.

В приложении Б представлен код программы с точками измерения полного времени, а в приложении В – с точками для измерения времени выполнения ФУ. Как видно, точки расставлены с целью профилирования основной процедуры программы.

Результаты профилирования представлены соответственно на рис. 3 и рис. 4. Как можно заметить, время выполнения в этих случаях практически не отличается – отличие составляет 1.7%, что можно списать на затраты при большем количестве измерений в случае с ФУ.

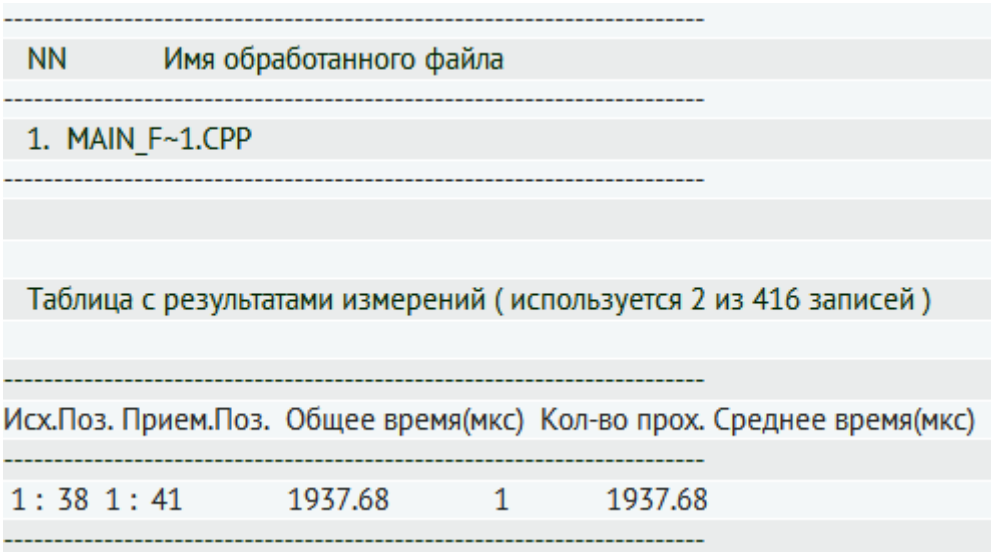

Рисунок 3 - Результаты профилирования ЛР1 (полное время)

NN Имя обработанного файла

1. MAIN_FU.CPP

Таблица с результатами измерений (используется 10 из 416 записей)

Исх.Поз.	Прием.Поз.	Общее время(мкс)	Кол-во прох.	Среднее время(мкс)
1 : 12	1 : 15	833.07	13	64.08
1 : 15	1 : 27	20.95	13	1.61
1 : 21	1 : 24	0.84	1	0.84
1 : 24	1 : 12	122.36	13	9.41
1 : 27	1 : 45	250.59	13	19.28
1 : 45	1 : 48	217.07	13	16.70
1 : 48	1 : 24	408.15	12	34.01
1 : 48	1 : 51	16.76	1	16.76
1 : 51	1 : 59	1.68	1	1.68
1 : 56	1 : 21	57.83	1	57.83

Рисунок 4 - Результаты профилирования ЛР1 (время ФУ)

Как можно отметить по результатам профилировки ФУ, наибольшее время занимает непосредственно вычисление. Можно попытаться сократить расходы путем замены обращения к переменным по указателю на обращение к глобальным переменным. Результат профилирования обновленной программы представлен на рис. 5 и 6.

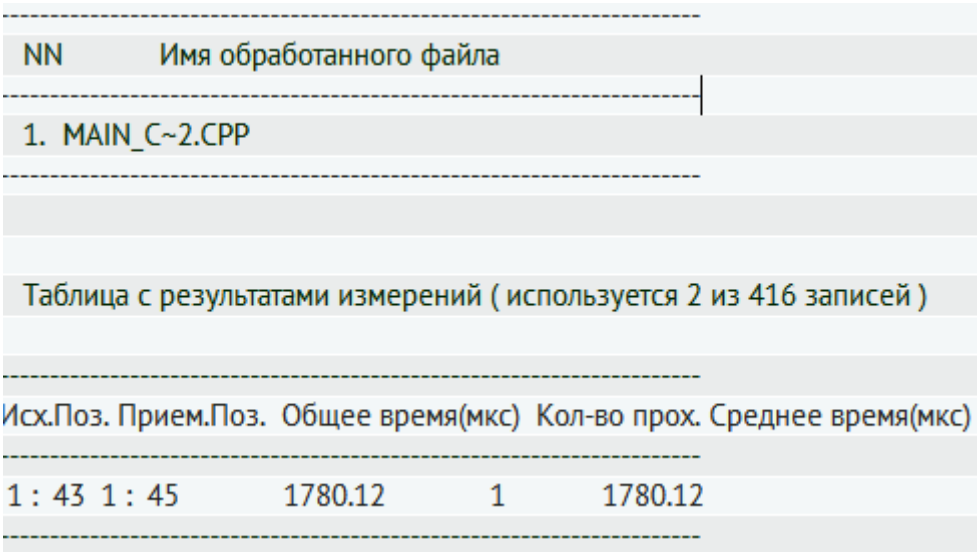

Рисунок 5 - Результаты профилирования обновленной ЛР1 (полное время)

NN Имя обработанного файла

1. MAIN_C~1.CPP

Таблица с результатами измерений (используется 10 из 416 записей)

Исх.Поз.	Прием.Поз.	Общее время(мкс)	Кол-во прох.	Среднее время(мкс)
1 : 18	1 : 21	842.29	13	64.79
1 : 21	1 : 32	20.95	13	1.61
1 : 26	1 : 29	0.84	1	0.84
1 : 29	1 : 18	98.06	13	7.54
1 : 32	1 : 50	265.68	13	20.44
1 : 50	1 : 53	186.90	13	14.38
1 : 53	1 : 29	417.37	12	34.78
1 : 53	1 : 56	17.60	1	17.60
1 : 56	1 : 63	1.68	1	1.68
1 : 61	1 : 26	2.51	1	2.51

Рисунок 6 - Результаты профилирования обновленной ЛР1 (время ФУ)

Как можно заметить оптимизация прошла успешно. Время выполнения программы уменьшилось как в случае полного времени, так и для каждого ФУ в отдельности. Время сократилось довольно ощутимо (5% в среднем), однако использование глобальных переменных снижает читабельность программы, а значит такой способ оптимизации не является панацеей.

Выводы.

В ходе выполнения данной работы был изучен профилировщик SAMPLER, на практике проведены вычисления профилей программ. Также было проанализировано полное время работы программы и время выполнения ФУ. Проведена успешная попытка частичной оптимизации.

ПРИЛОЖЕНИЕ А

ИЗМЕНЕННЫЕ ТЕСТОВЫЕ ПРОГРАММЫ

TEST_CYC.CPP:

```
1. #include "Sampler.h"
2. #define Size 10000
3. int i, tmp, dim[Size];
4.
5. void main()
6. {
7.   SAMPLE;
8.   for(i=0;i<Size/10;i++){ tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp; };
9.   SAMPLE;
10.  for(i=0;i<Size/5;i++){ tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp; };
11.  SAMPLE;
12.  for(i=0;i<Size/2;i++){ tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp; };
13.  SAMPLE;
14.  for(i=0;i<Size;i++)  { tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp; };
15.  SAMPLE;
16.  for(i=0;i<Size/10;i++)
17.  { tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp; };
18.  SAMPLE;
19.  for(i=0;i<Size/5;i++)
20.  { tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp; };
21.  SAMPLE;
22.  for(i=0;i<Size/2;i++)
23.  { tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp; };
24.  SAMPLE;
25.  for(i=0;i<Size;i++)
26.  { tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp; };
27.  SAMPLE;
28.  for(i=0;i<Size/10;i++)
29.  { tmp=dim[0];
30.  dim[0]=dim[i];
31.  dim[i]=tmp;
32.  };
33.  SAMPLE;
34.  for(i=0;i<Size/5;i++)
35.  { tmp=dim[0];
36.  dim[0]=dim[i];
37.  dim[i]=tmp;
```

```
38.  };
39.  SAMPLE;
40.  for(i=0;i<Size/2;i++)
41.  { tmp=dim[0];
42.  dim[0]=dim[i];
43.  dim[i]=tmp;
44.  };
45.  SAMPLE;
46.  for(i=0;i<Size;i++)
47.  { tmp=dim[0];
48.  dim[0]=dim[i];
49.  dim[i]=tmp;
50.  };
51.  SAMPLE;
52.}
```

TEST_CYC.CPP:

```
53.#include "Sampler.h"
54.const unsigned Size = 1000;
55.
56.
57.void TestLoop(int nTimes)
58.  {
59.  static int TestDim[Size];
60.  int tmp;
61.  int iLoop;
62.
63.  while (nTimes > 0)
64.  {
65.      nTimes --;
66.
67.      iLoop = Size;
68.      while (iLoop > 0)
69.      {
70.          iLoop -- ;
71.          tmp = TestDim[0];
72.          TestDim[0] = TestDim[nTimes];
73.          TestDim[nTimes] = tmp;
74.      }
75.  }
76.} /* TestLoop */
```

```
77.
78.
79.void main()
80.{
81.    SAMPLE;
82.    TestLoop(Size / 10); // 100 * 1000  повторений
83.    SAMPLE;
84.    TestLoop(Size / 5);  // 200 * 1000  повторений
85.    SAMPLE;
86.    TestLoop(Size / 2);  // 500 * 1000  повторений
87.    SAMPLE;
88.    TestLoop(Size / 1);  // 1000* 1000  повторений
89.    SAMPLE;
90.}
```

ПРИЛОЖЕНИЕ Б

ИЗМЕНЕННЫЙ КОД ЛР1 ДЛЯ ПОЛНОГО ВРЕМЕНИ

```
1. #include <math.h>
2. #include "Sampler.h"
3.
4. const double tol = 1.0e-6,
5.              a = 18.19,
6.              b = -23180.0,
7.              c = -.8858,
8.              logp = -4.60517;
9.
10.double  x = 5.0,
11.       fx = .0,
12.       dfx = .0,
13.       dx = .0,
14.       x1 = .0;
15.
16.void func()
17.{
18.    fx = a + b / x + c * log(x) - logp;
19.    dfx = -b / (x * x) + c / x;
20.}
21.
22.void newton()
23.{
24.    do
25.    {
26.        x1 = x;
27.        func();
28.        if(fabs(dfx) < tol)
29.        {
30.            if(dfx >= .0)
31.              dfx = tol;
32.            else
33.              dfx = -tol;
34.        }
35.        dx = fx / dfx;
36.        x = x1 - dx;
37.    }
38.    while(fabs(dx) >= fabs(tol * x));
```

```
39.}
40.
41.int main()
42.{
43.    SAMPLE;
44.    newton();
45.    SAMPLE;
46.    return 0;
47.}
```

ПРИЛОЖЕНИЕ В

ИЗМЕНЕННЫЙ КОД ЛР1 ДЛЯ ВРЕМЕНИ ФУ

```
1. #include <math.h>
2. #include "Sampler.h"
3.
4. const double tol = 1.0e-6,
5.              a = 18.19,
6.              b = -23180.0,
7.              c = -.8858,
8.              logp = -4.60517;
9.
10.double  x = 5.0,
11.       fx = .0,
12.       dfx = .0,
13.       dx = .0,
14.       x1 = .0;
15.
16.void func()
17.{
18.    SAMPLE;
19.    fx = a + b / x + c * log(x) - logp;
20.    dfx = -b / (x * x) + c / x;
21.    SAMPLE;
22.}
23.
24.void newton()
25.{
26.    SAMPLE;
27.    do
28.    {
29.        SAMPLE;
30.        x1 = x;
31.        func();
32.        SAMPLE;
33.        if(fabs(dfx) < tol)
34.        {
35.            SAMPLE;
36.            if(dfx >= .0)
37.            {
38.               SAMPLE;
```

```
39.             dfx = tol;
40.             SAMPLE;
41.          }
42.          else
43.          {
44.             SAMPLE;
45.             dfx = -tol;
46.             SAMPLE;
47.          }
48.          SAMPLE;
49.       }
50.       SAMPLE;
51.       dx = fx / dfx;
52.       x = x1 - dx;
53.       SAMPLE;
54.    }
55.    while(fabs(dx) >= fabs(tol * x));
56.    SAMPLE;
57.}
58.
59.int main()
60.{
61.    SAMPLE:
62.    newton();
63.    SAMPLE:
64.    return 0;
65.}
```