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

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 _____ Кошманов Н.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

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.

   Замечание: следует внимательно подойти к выбору ФУ для получения хороших результатов профилирования.
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

1. Использована старая версия SAMPLER в DOSBOX. Компиляция программ производилась в TurboC++ v3.1
2. В программах TEST_CYC.CPP и TEST_SUB.CPP расставлены контрольные точки. Коды программ представлены в Приложении А, Б соответственно.
3. Запущен SAMPLER для скомпилированных программ. Листинги представлены ниже.

TEST_CYC.EXE

```
---------------------------------------------------------------------
NN            Имя обработанного файла
---------------------------------------------------------------------
1.     ..\SAMPLE~1\SAMP16\TEST_S~1.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             4335.47             1             4335.47
-------------------------------------------------------------------------------
1 :   9  1 :  11             8669.27             1             8669.27
-------------------------------------------------------------------------------
1 :  11  1 :  13            21677.37             1            21677.37
-------------------------------------------------------------------------------
1 :  13  1 :  15            43348.87             1            43348.87
-------------------------------------------------------------------------------
1 :  15  1 :  18             4335.47             1             4335.47
-------------------------------------------------------------------------------
1 :  18  1 :  21             8670.95             1             8670.95
-------------------------------------------------------------------------------
1 :  21  1 :  24            21673.18             1            21673.18
-------------------------------------------------------------------------------
1 :  24  1 :  27            43348.03             1            43348.03
-------------------------------------------------------------------------------
1 :  27  1 :  34             4335.47             1             4335.47
-------------------------------------------------------------------------------
1 :  34  1 :  41             8669.27             1             8669.27
-------------------------------------------------------------------------------
1 :  41  1 :  48            21677.37             1            21677.37
-------------------------------------------------------------------------------
1 :  48  1 :  55            43343.00             1            43343.00
-------------------------------------------------------------------------------
```

Как видно увеличение общего времени с 7 по 13, с 15 по 24 и с 27 по 48 происходит линейно прямо пропорционально количеству итераций в цикле.

TEST_SUB.EXE

```
-----------------------------------------------------------------------
   NN            Имя обработанного файла
-----------------------------------------------------------------------
   1.  ..\SAMPLE~1\SAMP16\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 :  28  1 :  30           433699.02             1          433699.02
-----------------------------------------------------------------------
 1 :  30  1 :  32           867418.16             1          867418.16
-----------------------------------------------------------------------
 1 :  32  1 :  34          2168469.13             1         2168469.13
-----------------------------------------------------------------------
 1 :  34  1 :  36          4336950.84             1         4336950.84
-----------------------------------------------------------------------
```

Как видно увеличение общего времени с 28 по 34 происходит линейно прямо пропорционально количеству итераций в цикле.

4. Профилирование программы из ЛР1.

   Программа из ЛР1 (Вариант 8) модифицирована для компилятора Turbo C++ v3.1 путем замены ввода-вывода данных на генерацию.

   Проведено профилирование полной версии программы (Приложение В) и отдельных участков программы (Приложение Г). Листинги представлены ниже

Полный замер

```
-----------------------------------------------------------------------
   NN            Имя обработанного файла
-----------------------------------------------------------------------
   1.  ..\SAMPLE~1\SAMP16\8_MAIN~1.CPP
-----------------------------------------------------------------------

   Таблица с результатами измерений ( используется 2 из 416 записей )
```

Исх.Поз.	Прием.Поз.	Общее время(мкс)	Кол-во прох.	Среднее время(мкс)
1 : 52	1 : 55	1673.95	1	1673.95

Отдельные замеры

NN	Имя обработанного файла
1.	..\SAMPLE~1\SAMP16\8_ALL_~1.CPP

Таблица с результатами измерений (используется 27 из 416 записей)

Исх.Поз.	Прием.Поз.	Общее время(мкс)	Кол-во прох.	Среднее время(мкс)
1 : 15	1 : 17	2.51	1	2.51
1 : 17	1 : 19	71.24	5	14.25
1 : 19	1 : 21	10.06	5	2.01
1 : 21	1 : 23	171.81	10	17.18
1 : 23	1 : 21	8.38	5	1.68
1 : 23	1 : 25	8.38	5	1.68
1 : 25	1 : 27	79.62	5	15.92
1 : 27	1 : 17	8.38	4	2.10
1 : 27	1 : 29	1.68	1	1.68
1 : 29	1 : 33	7.54	1	7.54
1 : 33	1 : 35	2.51	1	2.51
1 : 35	1 : 37	8.38	3	2.79
1 : 37	1 : 39	86.32	6	14.39
1 : 39	1 : 41	17.60	6	2.93
1 : 41	1 : 43	564.04	30	18.80
1 : 43	1 : 49	22.63	15	1.51
1 : 43	1 : 45	18.44	15	1.23
1 : 45	1 : 47	182.71	15	12.18
1 : 47	1 : 49	10.06	15	0.67
1 : 49	1 : 41	61.18	24	2.55
1 : 49	1 : 51	13.41	6	2.23
1 : 51	1 : 53	5.87	3	1.96
1 : 51	1 : 37	5.87	3	1.96

```
-------------------------------------------------------------------------------
 1 :  53  1 :  55              63.70             3              21.23
-------------------------------------------------------------------------------
 1 :  55  1 :  57               6.70             3               2.23
-------------------------------------------------------------------------------
 1 :  57  1 :  59             268.19            15              17.88
-------------------------------------------------------------------------------
 1 :  59  1 :  57              29.33            12               2.44
 1 :  59  1 :  61               6.70             3               2.23
-------------------------------------------------------------------------------
 1 :  61  1 :  35               3.35             2               1.68
 1 :  61  1 :  63               1.68             1               1.68
-------------------------------------------------------------------------------
 1 :  63  1 :  79               2.51             1               2.51
-------------------------------------------------------------------------------
 1 :  67  1 :  76               1.68             1               1.68
-------------------------------------------------------------------------------
 1 :  76  1 :  15               5.03             1               5.03
-------------------------------------------------------------------------------
```

Как можно заметить, суммарное время выполнения программ из Приложений В, Г примерно равны.

$$2.51 + 71.24 + 10.06 + 171.81 + 8.38 + 8.38 + 79.62 + 1.68 + 7.54 + 2.51 + 8.38 + 86.32 + 17.60 + 564.04 + 22.63 + 18.44 + 182.71 + 10.06 + 61.18 + 13.41 + 5.87 + 5.87 + 63.70 + 6.70 + 268.19 + 29.33 + 6.70 + 3.35 + 1.68 + 2.51 + 1.68 + 5.03 = \mathbf{1749.11}$$

$$1673.95 \sim = 1749.11$$

Как видно из листинга условие 44 выполнилось 30 раз и которых 15 раз true, 15 раз false. Ветвление по переменной k, значение которой фиксировано в цикле по переменной i, можно вынести наружу, тем самым убрав проверку внутри цикла. Для этого введем переменную для суммирования в цикле, а затем присваивания суммы элементу массива после цикла. Измененный код программы представлен в Приложении Д. Листинг представлен ниже.

Листинг модифицированной программы.

```
-------------------------------------------------------------------------
   NN            Имя обработанного файла
-------------------------------------------------------------------------
   1.  ..\SAMPLE~1\SAMP16\8_MOD.CPP
-------------------------------------------------------------------------

   Таблица с результатами измерений ( используется 28 из 416 записей )

-------------------------------------------------------------------------
Исх.Поз. Прием.Поз.  Общее время(мкс)  Кол-во прох. Среднее время(мкс)
```

1 : 15	1 : 17	1.68	1	1.68
1 : 17	1 : 19	66.21	5	13.24
1 : 19	1 : 21	5.03	5	1.01
1 : 21	1 : 23	164.27	10	16.43
1 : 23	1 : 21	4.19	5	0.84
1 : 23	1 : 25	3.35	5	0.67
1 : 25	1 : 27	74.59	5	14.92
1 : 27	1 : 17	4.19	4	1.05
1 : 27	1 : 29	0.84	1	0.84
1 : 29	1 : 33	7.54	1	7.54
1 : 33	1 : 36	1.68	1	1.68
1 : 36	1 : 38	5.87	3	1.96
1 : 38	1 : 40	61.18	6	10.20
1 : 40	1 : 42	8.38	6	1.40
1 : 42	1 : 44	361.22	30	12.04
1 : 44	1 : 42	32.69	24	1.36
1 : 44	1 : 46	6.70	6	1.12
1 : 46	1 : 48	78.78	6	13.13
1 : 48	1 : 54	0.84	3	0.28
1 : 48	1 : 50	1.68	3	0.56
1 : 50	1 : 52	72.91	3	24.30
1 : 52	1 : 54	0.00	3	0.00
1 : 54	1 : 56	0.84	3	0.28
1 : 54	1 : 38	5.03	3	1.68
1 : 56	1 : 58	59.50	3	19.83
1 : 58	1 : 60	5.87	3	1.96
1 : 60	1 : 62	253.94	15	16.93
1 : 62	1 : 60	13.41	12	1.12
1 : 62	1 : 64	4.19	3	1.40
1 : 64	1 : 36	3.35	2	1.68
1 : 64	1 : 66	0.84	1	0.84
1 : 66	1 : 82	1.68	1	1.68
1 : 70	1 : 79	0.84	1	0.84
1 : 79	1 : 15	4.19	1	4.19

Как видно из листинга, количество вхождений в ветвление if по переменной k уменьшено с 30 до 6.

$$1.68 + 66.21 + 5.03 + 164.27 + 4.19 + 3.35 + 74.59 + 4.19 + 0.84 + 7.54 + 1.68 + 5.87 + 61.18 + 8.38 + 361.22 + 32.69 + 6.70 + 78.78 + 0.84 + 1.68 + 72.91 + 0.00 + 0.84 + 5.03 + 59.50 + 5.87 + 253.94 + 13.41 + 4.19 + 3.35 + 0.84 + 1.68 + 0.84 + 4.19 = \mathbf{1317.5}$$

$$1317.5 < 1749.11\ (\sim 300)$$

5. Профилирование вызовов модифицированной функции и её первоначальной версией.

   Создадим программу, в которой будет выполнено умножение матриц старым и новым алгоритмами (Приложение Е). Каждый вариант алгоритма вызовем по 3 раза, сравним результаты времени работы. Листинг представлен ниже.

Листинг сравнения алгоритмов

```
---------------------------------------------------------------------
   NN            Имя обработанного файла
---------------------------------------------------------------------
   1.  ..\SAMPLE~1\SAMP16\8_COMP.CPP
---------------------------------------------------------------------

   Таблица с результатами измерений ( используется 4 из 416 записей )

-------------------------------------------------------------------------------------
Исх.Поз. Прием.Поз.  Общее время(мкс)  Кол-во прох. Среднее время(мкс)
-------------------------------------------------------------------------------------
 1 :  75  1 :  78             3763.89             3            1254.63
-------------------------------------------------------------------------------------
 1 :  78  1 :  75                3.35             2               1.68
 1 :  78  1 :  88                3.35             1               3.35
-------------------------------------------------------------------------------------
 1 :  88  1 :  91             2777.45             3             925.82
-------------------------------------------------------------------------------------
 1 :  91  1 :  88                4.19             2               2.10
-------------------------------------------------------------------------------------
```

Как видно из листинга, среднее время выполнения модифицированного алгоритма ниже. Время вычленений уменьшилось примерно на 300 мкс, что соответствует результатам, предоставленным в предыдущем пункте.

Вывод

В ходе выполнения данной лабораторной работы было произведено вычисление профиля программы на C с помощью профилировщика Sampler. Произведена оптимизация программы из ЛР1.

ПРИЛОЖЕНИЕ А

Исходный код TEST_SYS.CPP с контрольными точками

```
1. #include "sampler.h"
2. #define Size 10000
3. int i, j, 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.    {
18.           tmp = dim[0]; dim[0] = dim[i]; dim[i] = tmp;
19.    };
20.    SAMPLE;
21.    for (i = 0; i < Size / 5; i++)
22.    {
23.           tmp = dim[0]; dim[0] = dim[i]; dim[i] = tmp;
24.    };
25.    SAMPLE;
26.    for (i = 0; i < Size / 2; i++)
27.    {
28.           tmp = dim[0]; dim[0] = dim[i]; dim[i] = tmp;
29.    };
30.    SAMPLE;
31.    for (i = 0; i < Size; i++)
32.    {
33.           tmp = dim[0]; dim[0] = dim[i]; dim[i] = tmp;
34.    };
35.    SAMPLE;
36.    for (i = 0; i < Size / 10; i++)
37.    {
38.           tmp = dim[0];
39.           dim[0] = dim[i];
40.    dim[i] = tmp;41.
       };
42.    SAMPLE;
43.    for (i = 0; i < Size / 5; i++)
44.    {
45.           tmp = dim[0];
46.           dim[0] = dim[i];
47.    dim[i] = tmp;48.
       };
49.    SAMPLE;
50.    for (i = 0; i < Size / 2; i++)
51.    {
52.           tmp = dim[0];
```

```
53.           dim[0] = dim[i];
54.    dim[i] = tmp;55.
       };
56.    SAMPLE;
57.    for (i = 0; i < Size; i++)
58.    {
59.           tmp = dim[0];
60.           dim[0] = dim[i];
61.    dim[i] = tmp;62.
       };
63.    SAMPLE;
64. }
```

ПРИЛОЖЕНИЕ Б

Исходный код TEST_SUB.CPP с контрольными точками

```
1.  #include "sampler.h"
2.  const unsigned Size = 1000;
3.
4.  void TestLoop(int nTimes)
5.  {
6.    static int TestDim[Size];
7.    int tmp;
8.    int iLoop;
9.
10.   while (nTimes > 0)
11.   {
12.     nTimes --;
13.
14.     iLoop = Size;
15.     while (iLoop > 0)
16.     {
17.       iLoop -- ;
18.       tmp = TestDim[0];
19.       TestDim[0] = TestDim[nTimes];
20.       TestDim[nTimes] = tmp;
21.     }
22.   }
23. } /* TestLoop */
24.
25.
26. void main()
27. {
28.    SAMPLE;
29.    TestLoop(Size / 10); // 100 * 1000  повторений
30.    SAMPLE;
31.    TestLoop(Size / 5);  // 200 * 1000  повторений
32.    SAMPLE;
33.    TestLoop(Size / 2);  // 500 * 1000  повторений
34.    SAMPLE;
35.    TestLoop(Size / 1);  // 1000* 1000  повторений
36.    SAMPLE;
37. }
```

ПРИЛОЖЕНИЕ В

Исходный код программы из ЛР1, полный замер

```
1.  #include <stdio.h>
2.  #include <sampler.h>
3.
4.  #define rmax 9
5.  #define cmax 3
6.
7.
8.  typedef double ary[rmax];
9.  typedef double arys[cmax];
10. typedef double ary2[rmax][cmax];
11. typedef double ary2s[cmax][cmax];
12.
13.
14. void get_data(ary2 x, ary y, int nrow, int ncol) {
15.     for (int i = 0; i < nrow; i++) {
16.     x[i][0] = 1;
17.             for (int j = 1; j < ncol; j++) {
18.                     x[i][j] = (i + 1) * x[i][j - 1];
19.             }
20.             y[i] = 2 * (i + 1);
21.     }
22. }
23.
24. void square(ary2 x, double* y, ary2s a, double* g, int nrow, int ncol) {
25.     for (int k = 0; k < ncol; k++) {
26.             for (int l = 0; l <= k; l++) {
27.                     a[k][l] = 0;
28.
29.                     for (int i = 0; i < nrow; i++) {
30.                             a[k][l] = a[k][l] + x[i][l] * x[i][k];
31.                             if (k != l)
32.                                     a[l][k] = a[k][l];
33.                     }
34.             }
35.
36.             g[k] = 0;
37.             for (int i = 0; i < nrow; i++) {
38.                     g[k] = g[k] + y[i] * x[i][k];
39.             }
40.     }
41. }
42.
43. int main() {
44.     int nrow = 5;
45.     int ncol = 3;
46.
47.     ary2 x;
48.     ary y;
49.
50.     arys g;
51.     ary2s a;
52.     SAMPLE;
53.     get_data(x, y, nrow, ncol);
54.     square(x, y, a, g, nrow, ncol);
55.     SAMPLE;
56.
57.     return 0;
58. }
```

ПРИЛОЖЕНИЕ Г

Исходный код программы из ЛР1, отдельные замеры

```
1.  #include <stdio.h>
2.  #include <sampler.h>
3.
4.  #define rmax 9
5.  #define cmax 3
6.
7.
8.  typedef double ary[rmax];
9.  typedef double arys[cmax];
10. typedef double ary2[rmax][cmax];
11. typedef double ary2s[cmax][cmax];
12.
13.
14. void get_data(ary2 x, ary y, int nrow, int ncol) {
15.   SAMPLE;
16.   for (int i = 0; i < nrow; i++) {
17.         SAMPLE;
18.         x[i][0] = 1;
19.         SAMPLE;
20.         for (int j = 1; j < ncol; j++) {
21.               SAMPLE;
22.               x[i][j] = (i + 1) * x[i][j - 1];
23.               SAMPLE;
24.         }
25.         SAMPLE;
26.         y[i] = 2 * (i + 1);
27.         SAMPLE;
28.   }
29.   SAMPLE;
30. }
31.
32. void square(ary2 x, double* y, ary2s a, double* g, int nrow, int ncol) {
33.   SAMPLE;
34.   for (int k = 0; k < ncol; k++) {
35.         SAMPLE;
36.         for (int l = 0; l <= k; l++) {
37.               SAMPLE;
38.               a[k][l] = 0;
39.               SAMPLE;
40.               for (int i = 0; i < nrow; i++) {
41.                     SAMPLE;
42.                     a[k][l] = a[k][l] + x[i][l] * x[i][k];
43.                     SAMPLE;
44.                     if (k != l) {
45.                           SAMPLE;
46.                           a[l][k] = a[k][l];
47.                           SAMPLE;
48.                     }
49.                     SAMPLE;
50.               }
51.               SAMPLE;
52.         }
53.         SAMPLE;
54.         g[k] = 0;
55.         SAMPLE;
56.         for (int i = 0; i < nrow; i++) {
57.               SAMPLE;
58.               g[k] = g[k] + y[i] * x[i][k];
59.               SAMPLE;
60.         }
```

```
61.         SAMPLE;
62.   }
63.   SAMPLE;
64. }
65.
66. int main() {
67.   SAMPLE;
68.   int nrow = 5;
69.   int ncol = 3;
70.
71.   ary2 x;
72.   ary y;
73.
74.   arys g;
75.   ary2s a;
76.   SAMPLE;
77.   get_data(x, y, nrow, ncol);
78.   square(x, y, a, g, nrow, ncol);
79.   SAMPLE;
80.
81.   return 0;
82. }
```

ПРИЛОЖЕНИЕ Д

Исходный код модифицированной программы из ЛР1

```
1.  #include <stdio.h>
2.  #include <sampler.h>
3.
4.  #define rmax 9
5.  #define cmax 3
6.
7.
8.  typedef double ary[rmax];
9.  typedef double arys[cmax];
10. typedef double ary2[rmax][cmax];
11. typedef double ary2s[cmax][cmax];
12.
13.
14. void get_data(ary2 x, ary y, int nrow, int ncol) {
15.   SAMPLE;
16.   for (int i = 0; i < nrow; i++) {
17.         SAMPLE;
18.         x[i][0] = 1;
19.         SAMPLE;
20.         for (int j = 1; j < ncol; j++) {
21.               SAMPLE;
22.               x[i][j] = (i + 1) * x[i][j - 1];
23.               SAMPLE;
24.         }
25.         SAMPLE;
26.         y[i] = 2 * (i + 1);
27.         SAMPLE;
28.   }
29.   SAMPLE;
30. }
31.
32. void square(ary2 x, double* y, ary2s a, double* g, int nrow, int ncol) {
33.   SAMPLE;
34.   double s;
35.   for (int k = 0; k < ncol; k++) {
36.         SAMPLE;
37.         for (int l = 0; l <= k; l++) {
38.               SAMPLE;
39.               s = 0;
40.               SAMPLE;
41.               for (int i = 0; i < nrow; i++) {
42.                     SAMPLE;
43.                     s = s + x[i][l] * x[i][k];
44.                     SAMPLE;
45.               }
46.               SAMPLE;
47.               a[k][l] = s;
48.               SAMPLE;
49.               if (k != l) {
50.                     SAMPLE;
51.                     a[l][k] = a[k][l];
52.                     SAMPLE;
53.               }
54.               SAMPLE;
55.         }
56.         SAMPLE;
57.         g[k] = 0;
58.         SAMPLE;
59.         for (int i = 0; i < nrow; i++) {
60.               SAMPLE;
```

```
61.                 g[k] = g[k] + y[i] * x[i][k];
62.                 SAMPLE;
63.         }
64.         SAMPLE;
65.   }
66.   SAMPLE;
67. }
68.
69. int main() {
70.   SAMPLE;
71.   int nrow = 5;
72.   int ncol = 3;
73.
74.   ary2 x;
75.   ary y;
76.
77.   arys g;
78.   ary2s a;
79.   SAMPLE;
80.   get_data(x, y, nrow, ncol);
81.   square(x, y, a, g, nrow, ncol);
82.   SAMPLE;
83.
84.   return 0;
85. }
```

ПРИЛОЖЕНИЕ Е

Исходный код программы сравнения

```
#include <stdio.h>
#include <sampler.h>

#define rmax 9
#define cmax 3


typedef double ary[rmax];
typedef double arys[cmax];
typedef double ary2[rmax][cmax];
typedef double ary2s[cmax][cmax];


void get_data(ary2 x, ary y, int nrow, int ncol) {
	for (int i = 0; i < nrow; i++) {
	x[i][0] = 1;
		for (int j = 1; j < ncol; j++) {
			x[i][j] = (i + 1) * x[i][j - 1];
		}
		y[i] = 2 * (i + 1);
	}
}

void mod_square(ary2 x, double* y, ary2s a, double* g, int nrow, int ncol) {
	double s;
	for (int k = 0; k < ncol; k++) {
		for (int l = 0; l <= k; l++) {
			s = 0;
			for (int i = 0; i < nrow; i++) {
				s = s + x[i][l] * x[i][k];
			}
			a[k][l] = s;
			if (k != l) {
				a[l][k] = a[k][l];
			}
		}
		g[k] = 0;
		for (int i = 0; i < nrow; i++) {
			g[k] = g[k] + y[i] * x[i][k];
		}
	}
}

void orig_square(ary2 x, double* y, ary2s a, double* g, int nrow, int ncol) {
	for (int k = 0; k < ncol; k++) {
		for (int l = 0; l <= k; l++) {
			a[k][l] = 0;

			for (int i = 0; i < nrow; i++) {
				a[k][l] = a[k][l] + x[i][l] * x[i][k];
				if (k != l)
					a[l][k] = a[k][l];
			}
		}

		g[k] = 0;
		for (int i = 0; i < nrow; i++) {
			g[k] = g[k] + y[i] * x[i][k];
		}
	}
```

```
61. }
62.
63. int main() {
64.
65.       int nrow = 5;
66.       int ncol = 3;
67.
68.       for (int i = 0; i < 3; i++){
69.             ary2 x;
70.             ary y;
71.
72.             arys g;
73.             ary2s a;
74.
75.             SAMPLE;
76.             get_data(x, y, nrow, ncol);
77.             orig_square(x, y, a, g, nrow, ncol);
78.             SAMPLE;
79.       }
80.
81.       for (int j = 0; j < 3; j++){
82.             ary2 x;
83.             ary y;
84.
85.             arys g;
86.             ary2s a;
87.
88.             SAMPLE;
89.             get_data(x, y, nrow, ncol);
90.             mod_square(x, y, a, g, nrow, ncol);
91.             SAMPLE;
92.       }
93.
94.       return 0;
95. }
```