**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 _____ Соколов И.Д.

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.

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.**

Был использован Sampler_old под DOSbox.

## 1. Результаты для тестовых программ

1.1. Test_CYC.EXE

| | | | | |
|---|---|---|---|---|
| 1 : 9 | 1 : 11 | 4337.15 | 1 | 4337.15 |
| 1 : 11 | 1 : 13 | 8670.11 | 1 | 8670.11 |
| 1 : 13 | 1 : 15 | 21679.04 | 1 | 21679.04 |
| 1 : 15 | 1 : 17 | 43343.84 | 1 | 43343.84 |
| 1 : 17 | 1 : 20 | 4341.34 | 1 | 4341.34 |
| 1 : 20 | 1 : 23 | 8670.95 | 1 | 8670.95 |
| 1 : 23 | 1 : 26 | 21672.34 | 1 | 21672.34 |
| 1 : 26 | 1 : 29 | 43349.70 | 1 | 43349.70 |
| 1 : 29 | 1 : 35 | 4336.31 | 1 | 4336.31 |
| 1 : 35 | 1 : 41 | 8670.11 | 1 | 8670.11 |
| 1 : 41 | 1 : 47 | 21678.20 | 1 | 21678.20 |
| 1 : 47 | 1 : 53 | 43343.00 | 1 | 43343.00 |

Участок 9-11 в ~10 раз меньше, 11-13 в ~5 раз меньше, 13-15 в ~2 раза меньше участка 15-17, что соответствует разнице в количестве итераций каждого из циклов.

1.2. TEST_SUB.EXE

```
 1 :  24  1 :  26          433699.02            1          433699.02
-------------------------------------------------------------------------
 1 :  26  1 :  28          867392.18            1          867392.18
-------------------------------------------------------------------------
 1 :  28  1 :  30         2168481.70            1         2168481.70
-------------------------------------------------------------------------
 1 :  30  1 :  32         4336951.68            1         4336951.68
-------------------------------------------------------------------------
```

Как и для TEST_CYC.EXE затраченное время на выполнение функции увеличивается во столько же раз, во сколько раз увеличивается количество итераций (приблизительно в 2, в 5 и в 10 раз соответственно).

## 2. Профилирование программы из ЛР1

3.1. Полный замер

Код для измерения выполнения всей программы приведен ниже:

```
1.  #include "math.h"
2.  #include "sampler.h"
3.
4.  float x, er, ec;
5.  unsigned char done;
6.
7.  float erf(float x)
8.    /* infinite series expansion of the Gaussian error function */
9.
10. {
11.   static const float sqrtpi = 1.7724538;
12.   static const float tol = 1.0E-4;
13.
14.   float x2, sum, sum1, term;
15.   int i;
16.
17.
18.   float erf_result;
19.   x2 = x * x;
20.   sum = x;
21.   term = x;
22.   i = 0;
```

```
23.       do {
24.         i = i + 1;
25.         sum1 = sum;
26.         term = 2.0 * term * x2 / (1.0 + 2.0 * i);
27.         sum = term + sum1;
28.       } while (term >= tol * sum);
29.       erf_result = 2.0 * sum * exp(-x2) / sqrtpi;
30.       return erf_result;
31.     }    /* erf */
32.
33.     float erfc(float x)
34.       /* complement of error function */
35.     {
36.       static const float sqrtpi = 1.7724538;
37.       int terms = 12;
38.
39.       float x2, u, v, sum;
40.       int i;
41.
42.       float erfc_result;
43.       x2 = x * x;
44.       v = 1.0 / (2.0 * x2);
45.       u = 1.0 + v * (terms + 1.0);
46.       for( i = terms; i >= 1; i --)
47.       {
48.         sum = 1.0 + i * v / u;
49.         u = sum;
50.       }
51.       erfc_result = exp(-x2) / (x * sum * sqrtpi);
52.       return erfc_result;
53.     }      /* ercf */
54.
55.     int main()
56.     {       /* main */
57.       SAMPLE;
58.       done = 0;
59.       x = 2;
60.       do {
```

```
61.       if (x < 0.0)
62.         done = 1;
63.       else
64.       {
65.         if (x == 0.0)
66.         {
67.           er = 0.0;
68.           ec = 1.0;
69.         }
70.         else if (x < 1.5)
71.         {
72.           er = erf(x);
73.           ec = 1.0 - er;
74.         }
75.         else
76.         {
77.           ec = erfc(x);
78.           er = 1.0 - ec;
79.         }  /* if */
80.         x = x - 1;
81.       }   /* if */
82.     } while (!done);
83.     SAMPLE;
84.     return 0;
85.   }
```

```
--------------------------------------------------------------------------------
 1 :  57  1 :  83             2156.42             1            2156.42
--------------------------------------------------------------------------------
```

Результат: 2156.42.

3.2. Отдельные замеры

Код для измерения выполнения функциональных участков программы приведен ниже:

```
1.  #include "math.h"
2.  #include "sampler.h"
3.
4.  float x, er, ec;
5.  unsigned char done;
6.
7.  float erf(float x)
8.    /* infinite series expansion of the Gaussian error function */
9.
10. {
11.   SAMPLE;
12.   static const float sqrtpi = 1.7724538;
13.   static const float tol = 1.0E-4;
14.
15.   float x2, sum, sum1, term;
16.   int i;
17.
18.
19.   float erf_result;
20.   x2 = x * x;
21.   sum = x;
22.   term = x;
23.   i = 0;
24.   do {
25.     SAMPLE;
26.     i = i + 1;
27.     sum1 = sum;
28.     term = 2.0 * term * x2 / (1.0 + 2.0 * i);
29.     sum = term + sum1;
30.     SAMPLE;
31.   } while (term >= tol * sum);
```

```
32.     erf_result = 2.0 * sum * exp(-x2) / sqrtpi;
33.     SAMPLE;
34.     return erf_result;
35.   }    /* erf */
36.
37.   float erfc(float x)
38.     /* complement of error function */
39.   {
40.     SAMPLE;
41.     static const float sqrtpi = 1.7724538;
42.     int terms = 12;
43.
44.     float x2, u, v, sum;
45.     int i;
46.
47.     float erfc_result;
48.     x2 = x * x;
49.     v = 1.0 / (2.0 * x2);
50.     u = 1.0 + v * (terms + 1.0);
51.     for( i = terms; i >= 1; i --)
52.     {
53.       SAMPLE;
54.       sum = 1.0 + i * v / u;
55.       u = sum;
56.       SAMPLE;
57.     }
58.     erfc_result = exp(-x2) / (x * sum * sqrtpi);
59.     SAMPLE;
60.     return erfc_result;
61.   }     /* ercf */
62.
63.   int main()
64.   {      /* main */
65.     SAMPLE;
```

```
66.     done = 0;
67.     x = 2;
68.     do {
69.       SAMPLE;
70.       if (x < 0.0)
71.       {
72.         SAMPLE;
73.         done = 1;
74.         SAMPLE;
75.       }
76.       else
77.       {
78.         SAMPLE;
79.         if (x == 0.0)
80.         {
81.           SAMPLE;
82.           er = 0.0;
83.           ec = 1.0;
84.           SAMPLE;
85.         }
86.         else if (x < 1.5)
87.         {
88.           SAMPLE;
89.           er = erf(x);
90.           ec = 1.0 - er;
91.           SAMPLE;
92.         }
93.         else
94.         {
95.           SAMPLE;
96.           ec = erfc(x);
97.           er = 1.0 - ec;
98.           SAMPLE;
99.         } /* if */
```

```
100.        x = x - 1;
101.        SAMPLE;
102.      }  /* if */
103.    } while (!done);
104.    SAMPLE;
105.    return 0;
106.  }
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | : | 11 | 1 | : | 25 | 72.91 | 1 | 72.91 |
| 1 | : | 25 | 1 | : | 30 | 306.74 | 7 | 43.82 |
| 1 | : | 30 | 1 | : | 25 | 118.17 | 6 | 19.70 |
| 1 | : | 30 | 1 | : | 33 | 254.78 | 1 | 254.78 |
| 1 | : | 33 | 1 | : | 91 | 124.88 | 1 | 124.88 |
| 1 | : | 40 | 1 | : | 53 | 297.52 | 1 | 297.52 |
| 1 | : | 53 | 1 | : | 56 | 212.04 | 12 | 17.67 |
| 1 | : | 56 | 1 | : | 53 | 27.66 | 11 | 2.51 |
| 1 | : | 56 | 1 | : | 59 | 234.67 | 1 | 234.67 |
| 1 | : | 59 | 1 | : | 98 | 125.71 | 1 | 125.71 |
| 1 | : | 65 | 1 | : | 69 | 2.51 | 1 | 2.51 |
| 1 | : | 69 | 1 | : | 78 | 101.41 | 3 | 33.80 |
| 1 | : | 69 | 1 | : | 72 | 5.87 | 1 | 5.87 |
| 1 | : | 72 | 1 | : | 74 | 1.68 | 1 | 1.68 |
| 1 | : | 74 | 1 | : | 104 | 3.35 | 1 | 3.35 |
| 1 | : | 78 | 1 | : | 95 | 158.40 | 1 | 158.40 |
| 1 | : | 78 | 1 | : | 88 | 10.06 | 1 | 10.06 |
| 1 | : | 78 | 1 | : | 81 | 5.87 | 1 | 5.87 |
| 1 | : | 81 | 1 | : | 84 | 2.51 | 1 | 2.51 |
| 1 | : | 84 | 1 | : | 101 | 5.03 | 1 | 5.03 |
| 1 | : | 88 | 1 | : | 11 | 54.48 | 1 | 54.48 |
| 1 | : | 91 | 1 | : | 101 | 5.87 | 1 | 5.87 |
| 1 | : | 95 | 1 | : | 40 | 55.31 | 1 | 55.31 |
| 1 | : | 98 | 1 | : | 101 | 88.00 | 1 | 88.00 |
| 1 | : | 101 | 1 | : | 69 | 10.06 | 3 | 3.35 |

Общее время выполнения различается приблизительно на 100 мкс (меньше, чем 5% от времени выполнения всей программы).

72.91 + 306.74 + 118.17 + 254.78 + 124.88 + 297.52 + 212.04 + 27.66 + 234.67 + 125.71 + 2.51 + 101.41 + 5.87 + 1.68 + 3.35 + 158.40 + 10.06 + 5.87 + 2.51 + 5.03 + 54.48 + 5.87 + 55.31 + 88.00 + 10.06 = 2285,49 — Суммарное время отдельных замеров

**4. Модификации**

4.1. Замена цикла do..while на for

4.2. Замена переменной done на проверу в условии for

4.3. Замена проверки x < 0 на проверку в условии for (что позволило убрать один оператор if..else)

4.4. Удаление проверки x == 0, и вынесение инициализации переменных er и ec в начало функции main.

4.5 Вынес переменную sqrtpi в глобальную область видимости, чтобы она отдельно не инициализировалась в каждой функции

4.6 Удаление переменной sum1, так как в нее присваивалось значение sum на каждой итерации цикла, но при этом не модифицировалось и использовалось в дальнейших вычислениях вместо sum.

4.7 Удаление переменно tol, так как она использовалась в одном месте

4.8 Удаление переменной sum из erfc по той же причине, что и в 4.6

4.9 Удаление переменной terms из erfc

4.10 Безусловно, если бы в компилятор поддерживал более современной стандарт, можно было бы поэкспериментировать с функциями erf и erfc из стандартной библиотеки (math.h).

Большое кол-во времени (~ 250 мкс) забирает на себя вычисление exp(-x2), но с этим трудно что-либо сделать.

В итоге код программы выглядел следующим образом:

```
1. #include "math.h"
2. #include "sampler.h"
3.
4. float x, er, ec;
5. unsigned char done;
6. float sqrtpi = 1.7724538;
7.
```

```
8.  float erf(float x)
9.   /* infinite series expansion of the Gaussian error function */
10.
11.  {
12.   float x2, sum, term;
13.   int i;
14.
15.
16.   float erf_result;
17.   x2 = x * x;
18.   sum = x;
19.   term = x;
20.   i = 0;
21.   do {
22.    i = i + 1;
23.    term = 2.0 * term * x2 / (1.0 + 2.0 * i);
24.    sum = term + sum;
25.   } while (term >= 1.0E-4 * sum);
26.   erf_result = 2.0 * sum * exp(-x2) / sqrtpi;
27.   return erf_result;
28.  }   /* erf */
29.
30.  float erfc(float x)
31.   /* complement of error function */
32.  {
33.   float u, v;
34.   int i;
35.
36.   float erfc_result;
37.   v = 1.0 / (2.0 * x * x);
38.   u = 1.0 + v * 13;
39.   for( i = 12; i >= 1; i --)
40.   {
41.    u = 1.0 + i * v / u;
```

```
42.     }
43.     erfc_result = exp(-x2) / (x * u * sqrtpi);
44.     return erfc_result;
45.   }     /* ercf */
46.
47.   int main()
48.   {     /* main */
49.     SAMPLE;
50.     er = 0.0;
51.     ec = 1.0;
52.     for (x = 1; x <= 2.0; x++) {
53.       if (x < 1.5) {
54.         er = erf(x);
55.         ec = 1.0 - er;
56.       }
57.       else {
58.         ec = erfc(x);
59.         er = 1.0 - ec;
60.       }
61.     }
62.     SAMPLE;
63.     return 0;
64.   }
```

Модифицированный код выигрывает по скорости около 146 мкс (при профилировании и модификации только функции main).

```
-------------------------------------------------------------------------
 1 :  57  1 :  70             2010.59             1             2010.59
-------------------------------------------------------------------------
```

После профилирования и модификации erf и erfc (выигрыш в ~203 мкс)

```
-------------------------------------------------------------------------
 1 :  50  1 :  63             1952.76             1             1952.76
-------------------------------------------------------------------------
```

**Выводы**

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