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_v2»

Студент гр. 8304 _____ Чешуин Д. И.

Преподаватель _____ Кирьянчиков В. А.

Санкт-Петербург
2022

Цель работы.

Изучить возможности измерения динамических характеристик программ с помощью профилировщиков на примере профилировщика SAMPLER_v2.

Задание.

1) Выполнить под управлением SAMPLER_v2 тестовые программы test_cyc.c и test_sub.c и привести отчет по результатам их выполнения с анализом параметров повторения циклов, структуры описания циклов, способов профилирования процедур и проверкой их влияния на точность и чувствительность профилирования.
2) Разработанную в лаб. работе 1 программу, реализующую заданный вычислительный алгоритм, разбить на функциональные участки (ФУ) и расставить на их границах контрольные точки (КТ) для выполнения с помощью ПИМ SAMPLER_v2 измерений и получения профиля выполнения программы, представляющего времена выполнения и количество выполнений каждого ФУ.
3) Скомпилировать полученную программу. При компиляции добавить путь к sampler.h в набор путей поиска включаемых файлов (`Isampler/libsampler` при компиляции, если архив был распакован в текущий каталог), при линковке добавить путь к libsampler.a в набор путей поиска библиотек и подключить её (флаги `-LSampler/build/libsampler -lsampler` при линковке).
4) Выполнить скомпилированную программу под управлением Sampler'а с внешним зацикливанием и получить отчет по результатам профилирования. Зацикливание можно выполнять при помощи программы sampler-repeat. Использование программы приведено в разделе 4 документа «Описание работы с ПИМ SAMPLER_v2». Число повторов зависит от сложности самой программы; имеет смысл начальное число запусков взять равным 10 и увеличивать его в 5–10 раз до тех пор, пока среднее время выполнения участков не стабилизируется, или на запуски станет уходить слишком много

времени, или на результаты станет уходить слишком много дискового пространства.

5) Проанализировать полученный отчет и выявить "узкие места", приводящие к ухудшению производительности программы.
6) Ввести в программу усовершенствования для повышения производительности, получить новые профили, добавить их в отчет и объяснить полученные результаты.

Ход работы.

Были выполнены под управлением монитора SAMPLER_v2 тестовые программы test_cyc.c и test_sub.c. Результаты представлены на рисунках 1 и 2.

исх	прием	общее время	кол-во проходов	среднее время
13	15	2888.997	1	2888.997
15	17	5777.562	1	5777.562
17	19	22685.790	1	22685.790
19	21	30288.517	1	30288.517
21	24	2838.786	1	2838.786
24	27	7433.506	1	7433.506
27	30	17894.892	1	17894.892
30	33	32811.390	1	32811.390
33	39	2711.913	1	2711.913
39	45	7466.908	1	7466.908
45	51	17196.182	1	17196.182
51	57	34344.333	1	34344.333

Рисунок 1 – Результат работы SAMPLER_v2 для программы test_cyc.c

исх	прием	общее время	кол-во проходов	среднее время
30	32	30517481.400	1	30517481.400
32	34	60805354.800	1	60805354.800
34	36	148895753.000	1	148895753.000
36	38	292465622.000	1	292465622.000

Рисунок 2 - Результат работы SAMPLER_v2 для программы test_sub.c

По результатам выполнения измерений видно, что время измерений растёт пропорционально количеству итераций циклов.

Была выполнена под управлением монитора SAMPLER_v2 программа из лабораторной работы №1. Результат измерений для полного времени выполнения функции *linfit* представлен на рисунке 3. Исходный код этой программы представлен в Приложении А.

исх	прием	общее время	кол-во проходов	среднее время
127	129	6659.047	1	6659.047

Рисунок 3 - Результат работы SAMPLER_v2 для измерения полного времени выполнения функции

Было выполнено разбиение программы из приложения А на функциональные участки. Исходный код программы, разбитый на функциональные участки, представлен в приложении Б. Полученные с помощью программы SAMPLER_v2 результаты представлены на рисунке 4.

исх	прием	общее время	кол-во проходов	среднее время
158	86	35.850	1	35.850
86	102	17.412	1	17.412
102	103	12.540	1	12.540
103	104	23.040	9	2.560
103	118	19.161	1	19.161
104	116	131.051	9	14.561
116	103	15.875	9	1.764
118	134	4.169	1	4.169
134	52	17.277	1	17.277
52	57	2.526	1	2.526
57	58	10.356	1	10.356
58	59	3.833	3	1.278
58	67	10.339	1	10.339
59	60	5.678	3	1.893
60	61	26.987	9	2.999
60	65	15.145	3	5.048
61	63	41.152	9	4.572
63	60	27.973	9	3.108
65	58	0.949	3	0.316
67	70	31.614	1	31.614
70	76	21.656	1	21.656
76	32	25.644	1	25.644
32	34	2.755	3	0.918
34	35	11.586	3	3.862
35	36	25.783	9	2.865
35	46	22.827	3	7.609
36	38	39.965	9	4.441
38	44	10.894	3	3.631
38	40	15.979	6	2.663
44	35	11.236	9	1.248
46	48	73.747	3	24.582
48	32	33.288	2	16.644
48	80	9.594	1	9.594
40	42	34.071	6	5.679
42	44	7.647	6	1.275
80	82	1.758	1	1.758
82	136	32.227	1	32.227
136	137	11.818	1	11.818
137	138	19.242	9	2.138
137	143	8.264	1	8.264
138	141	6440.486	9	715.610
141	137	23.862	9	2.651
143	145	174.133	1	174.133
145	160	26.770	1	26.770

Рисунок 4 - Результат работы SAMPLER_v2 для измерения полного времени выполнения функции, разбитой на функциональные участки

В итоге общее время выполнения – 7511,39 мкс. Разницу в 852,357 мкс с измерением для полного времени можно объяснить затратами на вызов функции измерения.

Как видно из результатов измерения времени выполнения функциональных участков – наиболее затратным фрагментом является тело цикла for (строчки 138-141).

Для решения этой проблемы в 139-140 и 144 строчках функция возведения в степень была заменена на произведение чисел, для упрощения расчетов была введена временная переменная *tmp*.

Была выполнена проверка изменённой программы. Результат представлен на рисунке 5. Исходный код модифицированной программы представлен в Приложении В.

исх	прием	общее время	кол-во проходов	среднее время
159	86	52.881	1	52.881
86	102	15.723	1	15.723
102	103	17.930	1	17.930
103	104	18.114	9	2.013
103	118	4.623	1	4.623
104	116	144.296	9	16.033
116	103	-0.522	9	-0.058
118	134	30.100	1	30.100
134	52	13.989	1	13.989
52	57	-0.528	1	-0.528
57	58	11.469	1	11.469
58	59	5.071	3	1.690
58	67	4.255	1	4.255
59	60	4.396	3	1.465
60	61	29.453	9	3.273
60	65	11.407	3	3.802
61	63	33.044	9	3.672
63	60	3.061	9	0.340
65	58	4.483	3	1.494
67	70	30.713	1	30.713
70	76	21.189	1	21.189
76	32	11.678	1	11.678
32	34	7.703	3	2.568
34	35	11.732	3	3.911
35	36	32.896	9	3.655
35	46	19.207	3	6.402
36	38	41.501	9	4.611
38	44	11.816	3	3.939
38	40	15.784	6	2.631
44	35	12.573	9	1.397
46	48	56.037	3	18.679
48	32	29.912	2	14.956
48	80	11.238	1	11.238
40	42	32.586	6	5.431
42	44	23.903	6	3.984
80	82	3.310	1	3.310
82	136	21.125	1	21.125
136	137	20.045	1	20.045
137	138	15.761	9	1.751
137	144	10.184	1	10.184
138	142	172.876	9	19.208
142	137	25.924	9	2.880
144	146	107.791	1	107.791
146	161	25.514	1	25.514

Рисунок 5 - Результат работы SAMPLER_v2 для измерения полного времени выполнения оптимизированной функции, разбитой на функциональные участки

В результате внесённых изменений удалось существенно снизить время выполнения программы. Общее время выполнения – 1176,243 мкс. Удалось добиться снижения времени выполнения на 6335,147 мкс (84%).

Выводы.

В ходе выполнения лабораторной работы были изучены возможности измерения динамических характеристик программ с помощью профилировщиков на примере профилировщика SAMPLER_v2. Для программы, взятой из первой лабораторной работы, было выполнено измерение времени работы, с последующим выявлением узкого места и его устранения – в результате чего удалось добиться существенно более быстрого выполнения программы.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include "sampler.h"
5
6 #define MAXR 9
7 #define MAXC 3
8
9 void get_data(double x[MAXR], double y[MAXR]) {
10  int i;
11  for (i = 0; i < MAXR; i++) {
12       x[i] = (double)i + 1;
13  }
14  y[0] = 2.07;
15  y[1] = 8.6;
16  y[2] = 14.42;
17  y[3] = 15.8;
18  y[4] = 18.92;
19  y[5] = 17.96;
20  y[6] = 12.98;
21  y[7] = 6.45;
22  y[8] = 0.27;
23 }
24
25 double deter(double a[MAXC][MAXC]) {
26  return(a[0][0] * (a[1][1] * a[2][2] - a[2][1] * a[1][2])
27       - a[0][1] * (a[1][0] * a[2][2] - a[2][0] * a[1][2])
28       + a[0][2] * (a[1][0] * a[2][1] - a[2][0] * a[1][1]));
29 }
30
31 void setup(double a[MAXC][MAXC], double b[MAXC][MAXC], double
y[MAXC], double coef[MAXC], int j, double det) {
32  int i;
33  for (i = 0; i < MAXC; i++) {
34       b[i][j] = y[i];
35       if (j > 0) {
36            b[i][j - 1] = a[i][j - 1];
37       }
38  }
39  coef[j] = deter(b) / det;
40 }
41
42  void  solve(double  a[MAXC][MAXC],  double  y[MAXC],  double
coef[MAXC]) {
43  double b[MAXC][MAXC];
44  int i, j;
45  double det;
46
47  for (i = 0; i < MAXC; i++) {
48       for (j = 0; j < MAXC; j++) {
49            b[i][j] = a[i][j];
```

```
50        }
51   }
52
53   det = deter(b);
54   if (det == 0) {
55        return;
56   }
57   else {
58        setup(a, b, y, coef, 0, det);
59        setup(a, b, y, coef, 1, det);
60        setup(a, b, y, coef, 2, det);
61   }
62 }
63
64   void   linfit(double   x[MAXR],   double   y[MAXR],   double
y_calc[MAXR], double coef[MAXC], double* corel_coef) {
65   double sum_x, sum_y, sum_xy, sum_x2, sum_y2;
66   double xi, yi, x2, sum_x3, sum_x4, sum_2y, srs;
67   int i;
68   double a[MAXC][MAXC];
69   double g[MAXC];
70
71   sum_x = 0.0;
72   sum_y = 0.0;
73   sum_xy = 0.0;
74   sum_x2 = 0.0;
75   sum_y2 = 0.0;
76   sum_x3 = 0.0;
77   sum_x4 = 0.0;
78   sum_2y = 0.0;
79
80   for (i = 0; i < MAXR; i++) {
81        xi = x[i];
82        yi = y[i];
83        x2 = xi * xi;
84        sum_x = sum_x + xi;
85        sum_y = sum_y + yi;
86        sum_xy = sum_xy + xi * yi;
87        sum_x2 = sum_x2 + x2;
88        sum_y2 = sum_y2 + yi * yi;
89        sum_x3 = sum_x3 + xi * x2;
90        sum_x4 = sum_x4 + x2 * x2;
91        sum_2y = sum_2y + x2 * yi;
92   }
93
94   a[0][0] = MAXR;
95   a[1][0] = sum_x;
96   a[0][1] = sum_x;
97   a[2][0] = sum_x2;
98   a[0][2] = sum_x2;
99   a[1][1] = sum_x2;
100       a[2][1] = sum_x3;
101       a[1][2] = sum_x3;
```

```
      102       a[2][2] = sum_x4;
      103       g[0] = sum_y;
      104       g[1] = sum_xy;
      105       g[2] = sum_2y;
      106
      107       solve(a, g, coef);
      108       srs = 0.0;
      109
      110       for (i = 0; i < MAXR; i++) {
      111            y_calc[i] = coef[0] + coef[1] * x[i] + coef[2] *
pow(x[i], 2);
      112            srs += pow(y[i] - y_calc[i], 2);
      113       }
      114       *corel_coef = sqrt(1.0 - srs / (sum_y2 - pow(sum_y, 2) /
MAXR));
      115 }
      116
      117 int main(int argc, char **argv)
      118 {
      119       sampler_init(&argc, argv);
      120       double x[MAXR];
      121       double y[MAXR];
      122       double y_calc[MAXR];
      123       double coef[MAXC];
      124       double corel_coef;
      125
      126       get_data(x, y);
      127       SAMPLE;
      128       linfit(x, y, y_calc, coef, &corel_coef);
      129       SAMPLE;
      130       return 0;
      131 }
```

ПРИЛОЖЕНИЕ Б.

КОД ПРОГРАММЫ С РАЗДЕЛЕНИЕМ НА ФУ

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include "sampler.h"
5
6 #define MAXR 9
7 #define MAXC 3
8
9 void get_data(double x[MAXR], double y[MAXR]) {
10  int i;
11  for (i = 0; i < MAXR; i++) {
12       x[i] = (double)i + 1;
13  }
14  y[0] = 2.07;
15  y[1] = 8.6;
16  y[2] = 14.42;
17  y[3] = 15.8;
18  y[4] = 18.92;
19  y[5] = 17.96;
20  y[6] = 12.98;
21  y[7] = 6.45;
22  y[8] = 0.27;
23 }
24
25 double deter(double a[MAXC][MAXC]) {
26  return(a[0][0] * (a[1][1] * a[2][2] - a[2][1] * a[1][2])
27       - a[0][1] * (a[1][0] * a[2][2] - a[2][0] * a[1][2])
28       + a[0][2] * (a[1][0] * a[2][1] - a[2][0] * a[1][1]));
29 }
30
31 void setup(double a[MAXC][MAXC], double b[MAXC][MAXC], double
y[MAXC], double coef[MAXC], int j, double det) {
32  SAMPLE;
33  int i;
34  SAMPLE;
35  for (i = 0; SAMPLE, i < MAXC; i++) {
36       SAMPLE;
37       b[i][j] = y[i];
38       SAMPLE;
39       if (j > 0) {
40            SAMPLE;
41            b[i][j - 1] = a[i][j - 1];
42            SAMPLE;
43       }
44       SAMPLE;
45  }
46  SAMPLE;
47  coef[j] = deter(b) / det;
```

```
      48  SAMPLE;
      49 }
      50
      51  void  solve(double  a[MAXC][MAXC],  double  y[MAXC],  double
coef[MAXC]) {
      52  SAMPLE;
      53  double b[MAXC][MAXC];
      54  int i, j;
      55  double det;
      56
      57  SAMPLE;
      58  for (i = 0; SAMPLE, i < MAXC; i++) {
      59       SAMPLE;
      60       for (j = 0; SAMPLE, j < MAXC; j++) {
      61            SAMPLE;
      62            b[i][j] = a[i][j];
      63            SAMPLE;
      64       }
      65       SAMPLE;
      66  }
      67  SAMPLE;
      68
      69  det = deter(b);
      70  SAMPLE;
      71  if (det == 0) {
      72       SAMPLE;
      73       return;
      74  }
      75  else {
      76       SAMPLE;
      77       setup(a, b, y, coef, 0, det);
      78       setup(a, b, y, coef, 1, det);
      79       setup(a, b, y, coef, 2, det);
      80       SAMPLE;
      81  }
      82  SAMPLE;
      83 }
      84
      85   void   linfit(double   x[MAXR],   double   y[MAXR],   double
y_calc[MAXR], double coef[MAXC], double* corel_coef) {
      86  SAMPLE;
      87  double sum_x, sum_y, sum_xy, sum_x2, sum_y2;
      88  double xi, yi, x2, sum_x3, sum_x4, sum_2y, srs;
      89  int i;
      90  double a[MAXC][MAXC];
      91  double g[MAXC];
      92
      93  sum_x = 0.0;
      94  sum_y = 0.0;
      95  sum_xy = 0.0;
      96  sum_x2 = 0.0;
      97  sum_y2 = 0.0;
      98  sum_x3 = 0.0;
```

```
99  sum_x4 = 0.0;
100       sum_2y = 0.0;
101
102       SAMPLE;
103       for (i = 0; SAMPLE, i < MAXR; i++) {
104            SAMPLE;
105            xi = x[i];
106            yi = y[i];
107            x2 = xi * xi;
108            sum_x = sum_x + xi;
109            sum_y = sum_y + yi;
110            sum_xy = sum_xy + xi * yi;
111            sum_x2 = sum_x2 + x2;
112            sum_y2 = sum_y2 + yi * yi;
113            sum_x3 = sum_x3 + xi * x2;
114            sum_x4 = sum_x4 + x2 * x2;
115            sum_2y = sum_2y + x2 * yi;
116            SAMPLE;
117       }
118       SAMPLE;
119
120       a[0][0] = MAXR;
121       a[1][0] = sum_x;
122       a[0][1] = sum_x;
123       a[2][0] = sum_x2;
124       a[0][2] = sum_x2;
125       a[1][1] = sum_x2;
126       a[2][1] = sum_x3;
127       a[1][2] = sum_x3;
128       a[2][2] = sum_x4;
129       g[0] = sum_y;
130       g[1] = sum_xy;
131       g[2] = sum_2y;
132       srs = 0.0;
133
134       SAMPLE;
135       solve(a, g, coef);
136       SAMPLE;
137       for (i = 0; SAMPLE, i < MAXR; i++) {
138            SAMPLE;
139            y_calc[i] = coef[0] + coef[1] * x[i] + coef[2] *
pow(x[i], 2);
140            srs += pow(y[i] - y_calc[i], 2);
141            SAMPLE;
142       }
143       SAMPLE;
144       *corel_coef = sqrt(1.0 - srs / (sum_y2 -pow(sum_y, 2)) /
MAXR));
145       SAMPLE;
146  }
147
148  int main(int argc, char **argv)
149  {
```

```
150      sampler_init(&argc, argv);
151      double x[MAXR];
152      double y[MAXR];
153      double y_calc[MAXR];
154      double coef[MAXC];
155      double corel_coef;
156
157      get_data(x, y);
158      SAMPLE;
159      linfit(x, y, y_calc, coef, &corel_coef);
160      SAMPLE;
161      return 0;
162 }
```

ПРИЛОЖЕНИЕ В.

КОД ПРОГРАММЫ С ОПТИМИЗАЦИЯМИ

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include "sampler.h"
5
6 #define MAXR 9
7 #define MAXC 3
8
9 void get_data(double x[MAXR], double y[MAXR]) {
10  int i;
11  for (i = 0; i < MAXR; i++) {
12       x[i] = (double)i + 1;
13  }
14  y[0] = 2.07;
15  y[1] = 8.6;
16  y[2] = 14.42;
17  y[3] = 15.8;
18  y[4] = 18.92;
19  y[5] = 17.96;
20  y[6] = 12.98;
21  y[7] = 6.45;
22  y[8] = 0.27;
23 }
24
25 double deter(double a[MAXC][MAXC]) {
26  return(a[0][0] * (a[1][1] * a[2][2] - a[2][1] * a[1][2])
27       - a[0][1] * (a[1][0] * a[2][2] - a[2][0] * a[1][2])
28       + a[0][2] * (a[1][0] * a[2][1] - a[2][0] * a[1][1]));
29 }
30
31 void setup(double a[MAXC][MAXC], double b[MAXC][MAXC], double
y[MAXC], double coef[MAXC], int j, double det) {
32  SAMPLE;
33  int i;
34  SAMPLE;
35  for (i = 0; SAMPLE, i < MAXC; i++) {
36       SAMPLE;
37       b[i][j] = y[i];
38       SAMPLE;
39       if (j > 0) {
40            SAMPLE;
41            b[i][j - 1] = a[i][j - 1];
42            SAMPLE;
43       }
44       SAMPLE;
45  }
46  SAMPLE;
47  coef[j] = deter(b) / det;
```

```
      48  SAMPLE;
      49 }
      50
      51  void  solve(double  a[MAXC][MAXC],  double  y[MAXC],  double
coef[MAXC]) {
      52  SAMPLE;
      53  double b[MAXC][MAXC];
      54  int i, j;
      55  double det;
      56
      57  SAMPLE;
      58  for (i = 0; SAMPLE, i < MAXC; i++) {
      59       SAMPLE;
      60       for (j = 0; SAMPLE, j < MAXC; j++) {
      61            SAMPLE;
      62            b[i][j] = a[i][j];
      63            SAMPLE;
      64       }
      65       SAMPLE;
      66  }
      67  SAMPLE;
      68
      69  det = deter(b);
      70  SAMPLE;
      71  if (det == 0) {
      72       SAMPLE;
      73       return;
      74  }
      75  else {
      76       SAMPLE;
      77       setup(a, b, y, coef, 0, det);
      78       setup(a, b, y, coef, 1, det);
      79       setup(a, b, y, coef, 2, det);
      80       SAMPLE;
      81  }
      82  SAMPLE;
      83 }
      84
      85   void   linfit(double   x[MAXR],   double   y[MAXR],   double
y_calc[MAXR], double coef[MAXC], double* corel_coef) {
      86  SAMPLE;
      87  double sum_x, sum_y, sum_xy, sum_x2, sum_y2, tmp;
      88  double xi, yi, x2, sum_x3, sum_x4, sum_2y, srs;
      89  int i;
      90  double a[MAXC][MAXC];
      91  double g[MAXC];
      92
      93  sum_x = 0.0;
      94  sum_y = 0.0;
      95  sum_xy = 0.0;
      96  sum_x2 = 0.0;
      97  sum_y2 = 0.0;
      98  sum_x3 = 0.0;
```

```
 99  sum_x4 = 0.0;
100       sum_2y = 0.0;
101
102       SAMPLE;
103       for (i = 0; SAMPLE, i < MAXR; i++) {
104            SAMPLE;
105            xi = x[i];
106            yi = y[i];
107            x2 = xi * xi;
108            sum_x = sum_x + xi;
109            sum_y = sum_y + yi;
110            sum_xy = sum_xy + xi * yi;
111            sum_x2 = sum_x2 + x2;
112            sum_y2 = sum_y2 + yi * yi;
113            sum_x3 = sum_x3 + xi * x2;
114            sum_x4 = sum_x4 + x2 * x2;
115            sum_2y = sum_2y + x2 * yi;
116            SAMPLE;
117       }
118       SAMPLE;
119
120       a[0][0] = MAXR;
121       a[1][0] = sum_x;
122       a[0][1] = sum_x;
123       a[2][0] = sum_x2;
124       a[0][2] = sum_x2;
125       a[1][1] = sum_x2;
126       a[2][1] = sum_x3;
127       a[1][2] = sum_x3;
128       a[2][2] = sum_x4;
129       g[0] = sum_y;
130       g[1] = sum_xy;
131       g[2] = sum_2y;
132       srs = 0.0;
133
134       SAMPLE;
135       solve(a, g, coef);
136       SAMPLE;
137       for (i = 0; SAMPLE, i < MAXR; i++) {
138            SAMPLE;
139            y_calc[i] = x[i] * (coef[0] + coef[1] + coef[2] *
x[i]);
140            tmp = (y[i] - y_calc[i]);
141            srs += tmp * tmp;
142            SAMPLE;
143       }
144       SAMPLE;
145       *corel_coef = sqrt(1.0 - srs / (sum_y2 - sum_y * sum_y)
/ MAXR);
146       SAMPLE;
147 }
148
149 int main(int argc, char **argv)
```

```
150 {
151      sampler_init(&argc, argv);
152      double x[MAXR];
153      double y[MAXR];
154      double y_calc[MAXR];
155      double coef[MAXC];
156      double corel_coef;
157
158      get_data(x, y);
159      SAMPLE;
160      linfit(x, y, y_calc, coef, &corel_coef);
161      SAMPLE;
162      return 0;
163 }
```