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

ОТЧЁТ

по лабораторной работе №1
по дисциплине «Качество и метрология программного обеспечения»
Тема: Расчет метрических характеристик качества разработки программ по метрикам Холстеда

Студент гр. 6304 _____ Корытов П.В.
Преподаватель _____ Кирьянчиков В.А.

Санкт-Петербург
2020

1. Цель работы

Измерить и сравнить метрики Холстеда для программ для C, Pascal и ассемблере.

2. Постановка задачи

Для заданного варианта программы обработки данных, представленной на языке Паскаль, разработать вычислительный алгоритм и также варианты программ его реализации на языках программирования Си и Ассемблер. Добиться, чтобы программы на Паскале и Си были работоспособны и давали корректные результаты (это потребуется в дальнейшем при проведении с ними измерительных экспериментов). Для получения ассемблерного представления программы можно либо самостоятельно написать код на ассемблере, реализующий заданный алгоритм, либо установить опцию “Code generation/Generate assembler source” при компиляции текста программы, представленной на языке Си. Во втором случае в ассемблерном представлении программы нужно удалить директивы описаний и отладочные директивы, оставив только исполняемые операторы.

Примечание

В заданных на Паскале вариантах программ обработки данных важен только вычислительный алгоритм, реализуемый программой. Поэтому для получения более корректных оценок характеристик программ следует учитывать только вычислительные операторы и исключить операторы, обеспечивающие интерфейс с пользователем и выдачу текстовых сообщений.

В сути алгоритма, реализуемого программой, нужно разобраться достаточно хорошо для возможности внесения в программу модификаций, выполняемых в дальнейшем при проведении измерений и улучшении характеристик качества программы.

Для измеряемых версий программ в дальнейшем будет нужно исключить операции ввода данных с клавиатуры и вывода на печать, потребляющие основную долю ресурса времени при выполнении программы. Поэтому можно уже в этой работе предусмотреть соответствующие преобразования исходной программы.

Для каждой из разработанных программ (включая исходную программу на Паскале) определить следующие метрические характеристики (по Холстеду):

1. Измеримые характеристики программ:
    - число простых (отдельных) операторов, в данной реализации;
    - число простых (отдельных) операндов, в данной реализации;
    - общее число всех операторов в данной реализации;
    - общее число всех операндов в данной реализации;
    - число вхождений j-го оператора в тексте программы;
    - число вхождений j-го операнда в тексте программы;
    - словарь программы;
    - длину программы.

2. Расчетные характеристики программы:
    - длину программы;
    - реальный и потенциальный объемы программы;
    - уровень программы;
    - интеллектуальное содержание программы;
    - работу программиста;
    - время программирования;
    - уровень используемого языка программирования;
    - ожидаемое число ошибок в программе.

Для характеристик длина программы, уровень программы, время программирования следует рассчитать как саму характеристику, так и ее оценку.

Расчет характеристик программ и их оценок выполнить двумя способами:

1. вручную (с калькулятором) или с помощью одного из доступных средств математических вычислений EXCEL, MATHCAD или MATLAB. Для программы на Ассемблере возможен только ручной расчет характеристик.
   При ручном расчете, в отличие от программного, нужно учитывать только выполняемые операторы, а все описания не учитываются. Соответственно все символы («;», «=», переменные, цифры), входящие в описания, не учитываются.
2. с помощью программы автоматизации расчета метрик Холстеда (для Си- и Паскаль-версий программ), краткая инструкция по работе с которой приведена в файле user_guide.

Для варианта расчета с использованием программы автоматизации желательно провести анализ влияния учета тех или иных групп операторов исследуемой программы на вычисляемые характеристики за счет задания разных ключей запуска.

При настройке параметров (ключей) запуска программы автоматизации следует задать корректное значение числа внешних связей — 2 анализируемой программы (по умолчанию задается 5), совпадающее с используемым при ручном расчете.

Результаты расчетов представить в виде таблиц с текстовыми комментариями:

1. Паскаль. Ручной расчет:
    1.1. Измеримые характеристики
    1.2. Расчетные характеристики
2. Паскаль. Программный расчет:
    2.1. Измеримые характеристики
    2.2. Расчетные характеристики
3. Си. Ручной расчет:
    3.1. Измеримые характеристики
    3.2. Расчетные характеристики
4. Си. Программный расчет:
    4.1. Измеримые характеристики
    4.2. Расчетные характеристики
5. Ассемблер. Ручной расчет
    5.1. Измеримые характеристики
    5.2. Расчетные характеристики
6. Сводная таблица для трех языков

3. Ход работы

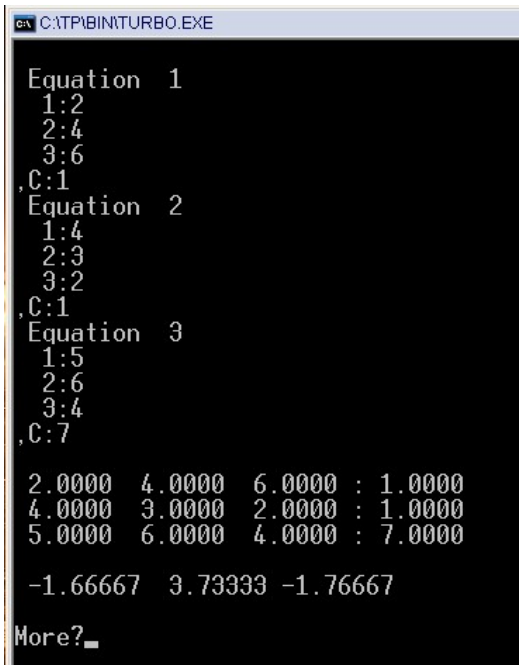
Вариант — 11, “Решение системы уравнений методом Крамера”.

3.1. Составление программ

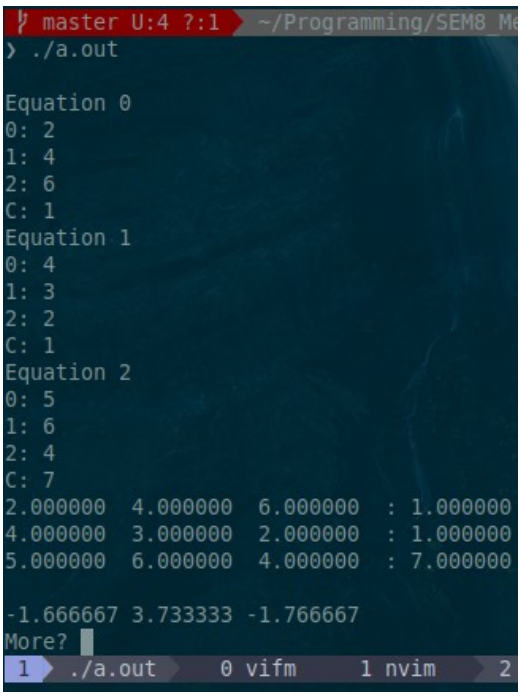
Код исходной программы на языке Pascal приведен в приложении А.

Разработан вариант программы на C (приложение В). Проведена отладка, чтобы добится одинаковых результатов в обоих программах (рис. 1).

С помощью команды:



(a) Запуск программы на Turbo Pascal



(b) Запуск программы на C

Рисунок 1 – Запуск программы

```
gcc -S prog.c
```

произведено создание программы на Ассемблере. Результат в приложении В.

3.2. Ручной расчёт метрик

Из программ на трех языках убраны операторы ввода-вывода и выделения памяти. Произведен подсчет операторов и операндов. Результаты расчёта в приложениях Г, Д, Е.

С помощью среды Jupyter Lab произведен расчёт ручной расчёт метрик. Данные в приложенных далее таблицах.

Таблица 1. Измеримые характеристики для языка Pascal

Метрика	Описание	Значение
η_1	Число простых (уникальных) операторов	16
η_2	Число простых (уникальных) операндов	13
N_1	Общее число всех операторов	93
N_2	Общее число всех операндов	84
η	Словарь	29
N	Опытная длина	177

Таблица 2. Расчётные характеристики для языка Pascal

Метрика	Описание	Значение
$N_{\text{теор}}$	Теоретическая длина	112.11
V	Объем	859.86
V^*	Потенциальный объем	11.61
L	Уровень	0.013502
I	Интеллектуальное содержание	16.63
E	Работа по программированию	63685.33
T	Время программирования	6368.53
λ	Уровень языка	0.16
B	Количество ошибок	1

Таблица 3. Измеримые характеристики для языка C

Метрика	Описание	Значение
η_1	Число простых (уникальных) операторов	16
η_2	Число простых (уникальных) операндов	13
N_1	Общее число всех операторов	118
N_2	Общее число всех операндов	103
η	Словарь	29
N	Опытная длина	221

Таблица 4. Расчётные характеристики для языка C

Метрика	Описание	Значение
$N_{\text{теор}}$	Теоретическая длина	112.11
V	Объем	1073.61
V^*	Потенциальный объем	11.61
L	Уровень	0.010814
I	Интеллектуальное содержание	16.94
E	Работа по программированию	99283.57
T	Время программирования	9928.36
λ	Уровень языка	0.13
B	Количество ошибок	2

Таблица 5. Измеримые характеристики для языка ассемблер

Метрика	Описание	Значение
η_1	Число простых (уникальных) операторов	52
η_2	Число простых (уникальных) операндов	29
N_1	Общее число всех операторов	418
N_2	Общее число всех операндов	519
η	Словарь	81
N	Опытная длина	937

Таблица 6. Расчётные характеристики для языка ассемблер

Метрика	Описание	Значение
$N_{\text{теор}}$	Теоретическая длина	437.30
V	Объем	5940.44
V^*	Потенциальый объем	11.61
L	Уровень	0.001954
I	Интеллектуальное содержание	12.77
E	Работа по программированию	3039613.58
T	Время программирования	303961.36
λ	Уровень языка	0.02
B	Количество ошибок	6

3.3. Программный расчёт

Проведен программный расчёт метрик с помощью программ. Логи работы в приложении Ж. Результаты в табличном виде:

Таблица 7. Измеримые характеристики для языка Pascal

Метрика	Описание	Значение
η_1	Число простых (уникальных) операторов	29
η_2	Число простых (уникальных) операндов	36
N_1	Общее число всех операторов	134
N_2	Общее число всех операндов	193
η	Словарь	65
N	Опытная длина	327

Таблица 8. Расчётные характеристики для языка Pascal

Метрика	Описание	Значение
$N_{\text{теор}}$	Теоретическая длина	327.00
V	Объем	1969.31
V^*	Потенциальый объем	11.61
L	Уровень	0.005895
I	Интеллектуальное содержание	25.33
E	Работа по программированию	334050
T	Время программирования	33405
λ	Уровень языка	0.07
B	Количество ошибок	2

Таблица 9. Измеримые характеристики для языка C

Метрика	Описание	Значение
η_1	Число простых (уникальных) операторов	16
η_2	Число простых (уникальных) операндов	13
N_1	Общее число всех операторов	118
N_2	Общее число всех операндов	103
η	Словарь	29
N	Опытная длина	221

Таблица 10. Расчётные характеристики для языка C

Метрика	Описание	Значение
$N_{\text{теор}}$	Теоретическая длина	307
V	Объем	3179.54
V^*	Потенциальый объем	11.61
L	Уровень	0.003651
I	Интеллектуальное содержание	20.96
E	Работа по программированию	870783
T	Время программирования	87078.30
λ	Уровень языка	0.04
B	Количество ошибок	4

Создана сводная таблица для всех языков (следующая страница).

Таблица 11. Сводная таблица

Метрика	Описание	Pascal (ручн.)	C (ручн.)	asm (ручн).	Pascal (прог.)	C (прог.)
η_1	Число простых (уникальных) операторов	16	16	52	29	35
η_2	Число простых (уникальных) операндов	13	13	29	36	27
N_1	Общее число всех операторов	93	118	418	134	300
N_2	Общее число всех операндов	84	103	519	193	234
η	Словарь	29	29	81	65	62
N	Опытная длина	177	221	937	327	534
$N_{\text{теор}}$	Теоретическая длина	112.11	112.11	437.30	327.00	307
V	Объем	859.86	1073.61	5940.44	1969.31	3179.54
V^*	Потенциальый объем	11.61	11.61	11.61	11.61	11.61
L	Уровень	0.013502	0.010814	0.001954	0.005895	0.003651
I	Интеллектуальное содержание	16.63	16.94	12.77	25.33	20.96
E	Работа по программированию	63685.33	99283.57	3039613.58	334050	870783
T	Время программирования	6368.53	9928.36	303961.36	33405	87078.30
λ	Уровень языка	0.16	0.13	0.02	0.07	0.04
B	Количество ошибок	1	2	6	2	4

4. Выводы

Проведен ручной и программный расчёт метрик Холстеда для одной и той же программы на языках C, Pascal и Ассемблер.

Установлен следующий порядок “уровней” программ: Pascal $>$ C $>$ Ассемблер. Pascal оценен выше C из-за операций динамического выделения памяти (`malloc`, `free`) в последнем. Ассемблер оценен существенно ниже обоих языков.

Результаты ручного и программного расчёта находятся в пределах одного порядка.

ПРИЛОЖЕНИЕ А

Код программы на Pascal

```
1  program simq1;
2  uses Crt;
3
4  const rmax = 3;
5        cmax = 3;
6
7  type arys = array[1..cmax] of real;
8       ary2s = array[1..rmax,1..cmax] of real;
9
10 var y,coef: arys;
11     a: ary2s;
12     n: integer;
13     yesno: char;
14     error: boolean;
15
16 procedure get_data(var a: ary2s;
17     var y: arys;
18     var n: integer);
19
20           { get the values for n, and arrays a,y }
21
22     var i,j: integer;
23
24 begin { procedure get_data }
25     writeln;
26     n:=rmax;
27     for i:=1 to n do
28     begin
29         writeln(' Equation',i:3);
30         for j:=1 to n do
31         begin
32             write(j:3,':');
```

```
33              read(a[i,j])
34          end;
35          write(',C:');
36          readln(y[i])
37      end;
38      writeln;
39      for i:=1 to n do
40      begin
41          for j:=1 to n do
42              write(a[i,j]:7:4,' ');
43          writeln(':',y[i]:7:4)
44      end;
45      writeln
46  end;      { procedure get_data }
47
48  procedure write_data;
49            { print out the answeres }
50
51      var i: integer;
52
53  begin     { write_data }
54      for i:=1 to n do
55          write(coef[i]:9:5);
56      writeln
57  end;      { write_data }
58
59  procedure solve(a: ary2s; y: arys;
60      var coef: arys; n: integer;
61      var error: boolean);
62
63  var
64      b: ary2s;
65      i,j: integer;
66      det: real;
```

```
67
68  function deter(a: ary2s): real;
69            { pascal program to calculate the determinant of a
              ↪  3-by-3matrix }
70
71  var
72      sum: real;
73
74      begin { function deter }
75      sum:=a[1,1]*(a[2,2]*a[3,3]-a[3,2]*a[2,3])
76      -a[1,2]*(a[2,1]*a[3,3]-a[3,1]*a[2,3])
77      +a[1,3]*(a[2,1]*a[3,2]-a[3,1]*a[2,2]);
78      deter:=sum
79  end;      { function deter }
80
81
82  procedure setup(var b: ary2s;
83      var coef: arys;
84      j: integer);
85
86      var i: integer;
87
88      begin { setup }
89      for i:=1 to n do
90      begin
91          b[i,j]:=y[i];
92          if j>1 then b[i,j-1]:=a[i,j-1]
93      end;
94      coef[j]:=deter(b)/det
95  end;      { setup }
96
97  begin     { procedure solve }
98  error:=false;
99  for i:=1 to n do
```

```
100       for j:=1 to n do
101           b[i,j]:=a[i,j];
102       det:=deter(b);
103       if det=0.0 then
104       begin
105           error:=true;
106           writeln(chr(7),'ERROR: matrix is singular.')
107       end
108       else
109       begin
110           setup(b,coef,1);
111           setup(b,coef,2);
112           setup(b,coef,3);
113       end   { else }
114   end;      { procedure solve }
115
116   begin     { MAIN program }
117   ClrScr;
118   writeln;
119   writeln('Simultaneous solution by Cramers rule');
120   repeat
121       get_data(a,y,n);
122       solve(a,y,coef,n,error);
123       if not error then write_data;
124       writeln;
125       write('More?');
126       readln(yesno);
127       ClrScr
128   until(yesno<>'Y')and(yesno<>'y')
129   end.
```

ПРИЛОЖЕНИЕ Б

Код программы на C

```c
#include ”stdio.h”
#include ”stdlib.h”
#include ”stdbool.h”

#define RMAX 3
#define CMAX 3

float** _alloc_matr(int a, int b) {
    float** m = (float**)malloc(a * sizeof(float*));
    for (int i = 0; i < CMAX; i ++) {
        m[i] = (float*)malloc(b * sizeof(float));
    }
    return m;
}

void _free_matr(float** m, int a) {
    for (int i = 0; i < a; i ++) {
        free(m[i]);
    }
    free(m);
}


/* print out the answers */
void print_matr(float** a, float* y) {
    for (int i = 0; i < RMAX; i++) {
        for (int j = 0; j < CMAX; j ++) {
            printf(”%f  ”, a[i][j]);
        }
        printf(”: %f\n”, y[i]);
    }
}
```

```
33
34  /* get the values for n, and arrays a,y */
35  void get_data(float** a, float* y) {
36      for (int i = 0; i < RMAX; i++) {
37          printf("Equation %d\n", i);
38          for (int j = 0; j < CMAX; j++) {
39              printf("%d: ", j);
40              scanf("%f", &a[i][j]);
41          }
42          printf("C: ");
43          scanf("%f", &y[i]);
44      }
45      print_matr(a, y);
46      printf("\n");
47  }
48
49  /* pascal program to calculate the determinant of a 3-by-3matrix */
50  float deter(float** a) {
51      return a[0][0] * (a[1][1] * a[2][2] - a [2][1] * a[1][2])
52           - a[0][1] * (a[1][0] * a[2][2] - a [2][0] * a[1][2])
53           + a[0][2] * (a[1][0] * a[2][1] - a [2][0] * a[1][1]);
54  }
55
56  void setup(float** a, float** b, float* coef, float* y, int j,
    ↪  float det) {
57      for (int i = 0; i < RMAX; i++) {
58          b[i][j] = y[i];
59          if (j > 0) {
60              b[i][j-1] = a[i][j-1];
61          }
62      }
63      coef[j] = deter(b) / det;
64  }
65
```

```
66  bool solve(float** a, float* y, float* coef) {
67      float** b = _alloc_matr(RMAX, CMAX);
68      float det = 0;
69      for (int i = 0; i < RMAX; i++) {
70          for (int j = 0; j < CMAX; j++) {
71              b[i][j] = a[i][j];
72          }
73      }
74      det = deter(b);
75      if (det == 0) {
76          printf("ERROR: matrix is singular.");
77          return true;
78      }
79      setup(a, b, coef, y, 0, det);
80      setup(a, b, coef, y, 1, det);
81      setup(a, b, coef, y, 2, det);
82      _free_matr(b, RMAX);
83      return false;
84  }
85
86  void write_data(float* coef) {
87      for (int i = 0; i < CMAX; i++) {
88          printf("%f ", coef[i]);
89      }
90      printf("\n");
91  }
92
93  int main() {
94      float** a = _alloc_matr(RMAX, CMAX);
95      float* y = (float*)malloc(CMAX * sizeof(float));
96      float* coef = (float*)malloc(CMAX * sizeof(float));
97      bool error;
98      char scan;
99      while (true) {
```

```
100         get_data(a, y);
101         error = solve(a, y, coef);
102         if (!error) {
103             write_data(coef);
104         }
105         printf("More? ");
106         scanf(" %c", &scan);
107         if (scan != 'y') {
108             break;
109         }
110     }
111     free(y);
112     free(coef);
113     _free_matr(a, RMAX);
114     return 0;
115 }
```

ПРИЛОЖЕНИЕ В

Код программы на ассемблере

```
1       .file   "prog.c"
2       .text
3       .globl   _alloc_matr
4       .type   _alloc_matr, @function
5   _alloc_matr:
6   .LFB5:
7       .cfi_startproc
8       pushq   %rbp
9       .cfi_def_cfa_offset 16
10      .cfi_offset 6, -16
11      movq    %rsp, %rbp
12      .cfi_def_cfa_register 6
13      pushq   %rbx
14      subq    $40, %rsp
15      .cfi_offset 3, -24
16      movl    %edi, -36(%rbp)
17      movl    %esi, -40(%rbp)
18      movl    -36(%rbp), %eax
19      cltq
20      salq    $3, %rax
21      movq    %rax, %rdi
22      call    malloc@PLT
23      movq    %rax, -24(%rbp)
24      movl    $0, -28(%rbp)
25      jmp     .L2
26  .L3:
27      movl    -40(%rbp), %eax
28      cltq
29      salq    $2, %rax
30      movl    -28(%rbp), %edx
31      movslq    %edx, %rdx
32      leaq    0(,%rdx,8), %rcx
```

```
33      movq    -24(%rbp), %rdx
34      leaq    (%rcx,%rdx), %rbx
35      movq    %rax, %rdi
36      call    malloc@PLT
37      movq    %rax, (%rbx)
38      addl    $1, -28(%rbp)
39  .L2:
40      cmpl    $2, -28(%rbp)
41      jle     .L3
42      movq    -24(%rbp), %rax
43      addq    $40, %rsp
44      popq    %rbx
45      popq    %rbp
46      .cfi_def_cfa 7, 8
47      ret
48      .cfi_endproc
49  .LFE5:
50      .size    _alloc_matr, .-_alloc_matr
51      .globl    _free_matr
52      .type    _free_matr, @function
53  _free_matr:
54  .LFB6:
55      .cfi_startproc
56      pushq    %rbp
57      .cfi_def_cfa_offset 16
58      .cfi_offset 6, -16
59      movq    %rsp, %rbp
60      .cfi_def_cfa_register 6
61      subq    $32, %rsp
62      movq    %rdi, -24(%rbp)
63      movl    %esi, -28(%rbp)
64      movl    $0, -4(%rbp)
65      jmp     .L6
66  .L7:
```

```
 67      movl    -4(%rbp), %eax
 68      cltq
 69      leaq    0(,%rax,8), %rdx
 70      movq    -24(%rbp), %rax
 71      addq    %rdx, %rax
 72      movq    (%rax), %rax
 73      movq    %rax, %rdi
 74      call    free@PLT
 75      addl    $1, -4(%rbp)
 76  .L6:
 77      movl    -4(%rbp), %eax
 78      cmpl    -28(%rbp), %eax
 79      jl    .L7
 80      movq    -24(%rbp), %rax
 81      movq    %rax, %rdi
 82      call    free@PLT
 83      nop
 84      leave
 85      .cfi_def_cfa 7, 8
 86      ret
 87      .cfi_endproc
 88  .LFE6:
 89      .size    _free_matr, .-_free_matr
 90      .section    .rodata
 91  .LC0:
 92      .string    "%f  "
 93  .LC1:
 94      .string    ": %f\n"
 95      .text
 96      .globl    print_matr
 97      .type    print_matr, @function
 98  print_matr:
 99  .LFB7:
100      .cfi_startproc
```

```
101     pushq   %rbp
102     .cfi_def_cfa_offset 16
103     .cfi_offset 6, -16
104     movq    %rsp, %rbp
105     .cfi_def_cfa_register 6
106     subq    $32, %rsp
107     movq    %rdi, -24(%rbp)
108     movq    %rsi, -32(%rbp)
109     movl    $0, -8(%rbp)
110     jmp     .L9
111 .L12:
112     movl    $0, -4(%rbp)
113     jmp     .L10
114 .L11:
115     movl    -8(%rbp), %eax
116     cltq
117     leaq    0(,%rax,8), %rdx
118     movq    -24(%rbp), %rax
119     addq    %rdx, %rax
120     movq    (%rax), %rax
121     movl    -4(%rbp), %edx
122     movslq    %edx, %rdx
123     salq    $2, %rdx
124     addq    %rdx, %rax
125     movss     (%rax), %xmm0
126     cvtss2sd    %xmm0, %xmm0
127     leaq    .LC0(%rip), %rdi
128     movl    $1, %eax
129     call    printf@PLT
130     addl    $1, -4(%rbp)
131 .L10:
132     cmpl    $2, -4(%rbp)
133     jle     .L11
134     movl    -8(%rbp), %eax
```

```
135     cltq
136     leaq    0(,%rax,4), %rdx
137     movq    -32(%rbp), %rax
138     addq    %rdx, %rax
139     movss    (%rax), %xmm0
140     cvtss2sd    %xmm0, %xmm0
141     leaq    .LC1(%rip), %rdi
142     movl    $1, %eax
143     call    printf@PLT
144     addl    $1, -8(%rbp)
145 .L9:
146     cmpl    $2, -8(%rbp)
147     jle    .L12
148     nop
149     leave
150     .cfi_def_cfa 7, 8
151     ret
152     .cfi_endproc
153 .LFE7:
154     .size    print_matr, .-print_matr
155     .section    .rodata
156 .LC2:
157     .string    "Equation %d\n"
158 .LC3:
159     .string    "%d: "
160 .LC4:
161     .string    "%f"
162 .LC5:
163     .string    "C: "
164     .text
165     .globl    get_data
166     .type    get_data, @function
167 get_data:
168 .LFB8:
```

```
169     .cfi_startproc
170     pushq    %rbp
171     .cfi_def_cfa_offset 16
172     .cfi_offset 6, -16
173     movq    %rsp, %rbp
174     .cfi_def_cfa_register 6
175     subq    $32, %rsp
176     movq    %rdi, -24(%rbp)
177     movq    %rsi, -32(%rbp)
178     movl    $0, -8(%rbp)
179     jmp    .L14
180 .L17:
181     movl    -8(%rbp), %eax
182     movl    %eax, %esi
183     leaq    .LC2(%rip), %rdi
184     movl    $0, %eax
185     call    printf@PLT
186     movl    $0, -4(%rbp)
187     jmp    .L15
188 .L16:
189     movl    -4(%rbp), %eax
190     movl    %eax, %esi
191     leaq    .LC3(%rip), %rdi
192     movl    $0, %eax
193     call    printf@PLT
194     movl    -8(%rbp), %eax
195     cltq
196     leaq    0(,%rax,8), %rdx
197     movq    -24(%rbp), %rax
198     addq    %rdx, %rax
199     movq    (%rax), %rax
200     movl    -4(%rbp), %edx
201     movslq    %edx, %rdx
202     salq    $2, %rdx
```

```
203     addq    %rdx, %rax
204     movq    %rax, %rsi
205     leaq    .LC4(%rip), %rdi
206     movl    $0, %eax
207     call    __isoc99_scanf@PLT
208     addl    $1, -4(%rbp)
209 .L15:
210     cmpl    $2, -4(%rbp)
211     jle    .L16
212     leaq    .LC5(%rip), %rdi
213     movl    $0, %eax
214     call    printf@PLT
215     movl    -8(%rbp), %eax
216     cltq
217     leaq    0(,%rax,4), %rdx
218     movq    -32(%rbp), %rax
219     addq    %rdx, %rax
220     movq    %rax, %rsi
221     leaq    .LC4(%rip), %rdi
222     movl    $0, %eax
223     call    __isoc99_scanf@PLT
224     addl    $1, -8(%rbp)
225 .L14:
226     cmpl    $2, -8(%rbp)
227     jle    .L17
228     movq    -32(%rbp), %rdx
229     movq    -24(%rbp), %rax
230     movq    %rdx, %rsi
231     movq    %rax, %rdi
232     call    print_matr
233     movl    $10, %edi
234     call    putchar@PLT
235     nop
236     leave
```

```
237     .cfi_def_cfa 7, 8
238     ret
239     .cfi_endproc
240 .LFE8:
241     .size    get_data, .-get_data
242     .globl    deter
243     .type    deter, @function
244 deter:
245 .LFB9:
246     .cfi_startproc
247     pushq    %rbp
248     .cfi_def_cfa_offset 16
249     .cfi_offset 6, -16
250     movq    %rsp, %rbp
251     .cfi_def_cfa_register 6
252     movq    %rdi, -8(%rbp)
253     movq    -8(%rbp), %rax
254     movq    (%rax), %rax
255     movss    (%rax), %xmm2
256     movq    -8(%rbp), %rax
257     addq    $8, %rax
258     movq    (%rax), %rax
259     addq    $4, %rax
260     movss    (%rax), %xmm1
261     movq    -8(%rbp), %rax
262     addq    $16, %rax
263     movq    (%rax), %rax
264     addq    $8, %rax
265     movss    (%rax), %xmm0
266     mulss    %xmm1, %xmm0
267     movq    -8(%rbp), %rax
268     addq    $16, %rax
269     movq    (%rax), %rax
270     addq    $4, %rax
```

```
271     movss    (%rax), %xmm3
272     movq    -8(%rbp), %rax
273     addq    $8, %rax
274     movq    (%rax), %rax
275     addq    $8, %rax
276     movss    (%rax), %xmm1
277     mulss    %xmm3, %xmm1
278     subss    %xmm1, %xmm0
279     mulss    %xmm2, %xmm0
280     movq    -8(%rbp), %rax
281     movq    (%rax), %rax
282     addq    $4, %rax
283     movss    (%rax), %xmm3
284     movq    -8(%rbp), %rax
285     addq    $8, %rax
286     movq    (%rax), %rax
287     movss    (%rax), %xmm2
288     movq    -8(%rbp), %rax
289     addq    $16, %rax
290     movq    (%rax), %rax
291     addq    $8, %rax
292     movss    (%rax), %xmm1
293     mulss    %xmm2, %xmm1
294     movq    -8(%rbp), %rax
295     addq    $16, %rax
296     movq    (%rax), %rax
297     movss    (%rax), %xmm4
298     movq    -8(%rbp), %rax
299     addq    $8, %rax
300     movq    (%rax), %rax
301     addq    $8, %rax
302     movss    (%rax), %xmm2
303     mulss    %xmm4, %xmm2
304     subss    %xmm2, %xmm1
```

```
305     mulss    %xmm3, %xmm1
306     subss    %xmm1, %xmm0
307     movaps    %xmm0, %xmm1
308     movq    -8(%rbp), %rax
309     movq    (%rax), %rax
310     addq    $8, %rax
311     movss    (%rax), %xmm3
312     movq    -8(%rbp), %rax
313     addq    $8, %rax
314     movq    (%rax), %rax
315     movss    (%rax), %xmm2
316     movq    -8(%rbp), %rax
317     addq    $16, %rax
318     movq    (%rax), %rax
319     addq    $4, %rax
320     movss    (%rax), %xmm0
321     mulss    %xmm2, %xmm0
322     movq    -8(%rbp), %rax
323     addq    $16, %rax
324     movq    (%rax), %rax
325     movss    (%rax), %xmm4
326     movq    -8(%rbp), %rax
327     addq    $8, %rax
328     movq    (%rax), %rax
329     addq    $4, %rax
330     movss    (%rax), %xmm2
331     mulss    %xmm4, %xmm2
332     subss    %xmm2, %xmm0
333     mulss    %xmm3, %xmm0
334     addss    %xmm1, %xmm0
335     popq    %rbp
336     .cfi_def_cfa 7, 8
337     ret
338     .cfi_endproc
```

```
339 .LFE9:
340     .size    deter, .-deter
341     .globl    setup
342     .type    setup, @function
343 setup:
344 .LFB10:
345     .cfi_startproc
346     pushq    %rbp
347     .cfi_def_cfa_offset 16
348     .cfi_offset 6, -16
349     movq    %rsp, %rbp
350     .cfi_def_cfa_register 6
351     subq    $56, %rsp
352     movq    %rdi, -24(%rbp)
353     movq    %rsi, -32(%rbp)
354     movq    %rdx, -40(%rbp)
355     movq    %rcx, -48(%rbp)
356     movl    %r8d, -52(%rbp)
357     movss    %xmm0, -56(%rbp)
358     movl    $0, -4(%rbp)
359     jmp    .L21
360 .L23:
361     movl    -4(%rbp), %eax
362     cltq
363     leaq    0(,%rax,4), %rdx
364     movq    -48(%rbp), %rax
365     addq    %rax, %rdx
366     movl    -4(%rbp), %eax
367     cltq
368     leaq    0(,%rax,8), %rcx
369     movq    -32(%rbp), %rax
370     addq    %rcx, %rax
371     movq    (%rax), %rax
372     movl    -52(%rbp), %ecx
```

```
373     movslq    %ecx, %rcx
374     salq    $2, %rcx
375     addq    %rcx, %rax
376     movss    (%rdx), %xmm0
377     movss    %xmm0, (%rax)
378     cmpl    $0, -52(%rbp)
379     jle    .L22
380     movl    -4(%rbp), %eax
381     cltq
382     leaq    0(,%rax,8), %rdx
383     movq    -24(%rbp), %rax
384     addq    %rdx, %rax
385     movq    (%rax), %rax
386     movl    -52(%rbp), %edx
387     movslq    %edx, %rdx
388     salq    $2, %rdx
389     subq    $4, %rdx
390     addq    %rax, %rdx
391     movl    -4(%rbp), %eax
392     cltq
393     leaq    0(,%rax,8), %rcx
394     movq    -32(%rbp), %rax
395     addq    %rcx, %rax
396     movq    (%rax), %rax
397     movl    -52(%rbp), %ecx
398     movslq    %ecx, %rcx
399     salq    $2, %rcx
400     subq    $4, %rcx
401     addq    %rcx, %rax
402     movss    (%rdx), %xmm0
403     movss    %xmm0, (%rax)
404 .L22:
405     addl    $1, -4(%rbp)
406 .L21:
```

```
407     cmpl    $2, -4(%rbp)
408     jle    .L23
409     movq    -32(%rbp), %rax
410     movq    %rax, %rdi
411     call    deter
412     movl    -52(%rbp), %eax
413     cltq
414     leaq    0(,%rax,4), %rdx
415     movq    -40(%rbp), %rax
416     addq    %rdx, %rax
417     divss    -56(%rbp), %xmm0
418     movss    %xmm0, (%rax)
419     nop
420     leave
421     .cfi_def_cfa 7, 8
422     ret
423     .cfi_endproc
424 .LFE10:
425     .size    setup, .-setup
426     .section    .rodata
427 .LC7:
428     .string    "ERROR: matrix is singular."
429     .text
430     .globl    solve
431     .type    solve, @function
432 solve:
433 .LFB11:
434     .cfi_startproc
435     pushq    %rbp
436     .cfi_def_cfa_offset 16
437     .cfi_offset 6, -16
438     movq    %rsp, %rbp
439     .cfi_def_cfa_register 6
440     subq    $64, %rsp
```

```
441     movq    %rdi, -40(%rbp)
442     movq    %rsi, -48(%rbp)
443     movq    %rdx, -56(%rbp)
444     movl    $3, %esi
445     movl    $3, %edi
446     call    _alloc_matr
447     movq    %rax, -8(%rbp)
448     pxor    %xmm0, %xmm0
449     movss    %xmm0, -12(%rbp)
450     movl    $0, -20(%rbp)
451     jmp    .L25
452 .L28:
453     movl    $0, -16(%rbp)
454     jmp    .L26
455 .L27:
456     movl    -20(%rbp), %eax
457     cltq
458     leaq    0(,%rax,8), %rdx
459     movq    -40(%rbp), %rax
460     addq    %rdx, %rax
461     movq    (%rax), %rax
462     movl    -16(%rbp), %edx
463     movslq    %edx, %rdx
464     salq    $2, %rdx
465     addq    %rax, %rdx
466     movl    -20(%rbp), %eax
467     cltq
468     leaq    0(,%rax,8), %rcx
469     movq    -8(%rbp), %rax
470     addq    %rcx, %rax
471     movq    (%rax), %rax
472     movl    -16(%rbp), %ecx
473     movslq    %ecx, %rcx
474     salq    $2, %rcx
```

```
475     addq    %rcx, %rax
476     movss    (%rdx), %xmm0
477     movss    %xmm0, (%rax)
478     addl    $1, -16(%rbp)
479 .L26:
480     cmpl    $2, -16(%rbp)
481     jle    .L27
482     addl    $1, -20(%rbp)
483 .L25:
484     cmpl    $2, -20(%rbp)
485     jle    .L28
486     movq    -8(%rbp), %rax
487     movq    %rax, %rdi
488     call    deter
489     movd    %xmm0, %eax
490     movl    %eax, -12(%rbp)
491     pxor    %xmm0, %xmm0
492     ucomiss    -12(%rbp), %xmm0
493     jp    .L29
494     pxor    %xmm0, %xmm0
495     ucomiss    -12(%rbp), %xmm0
496     jne    .L29
497     leaq    .LC7(%rip), %rdi
498     movl    $0, %eax
499     call    printf@PLT
500     movl    $1, %eax
501     jmp    .L31
502 .L29:
503     movl    -12(%rbp), %edi
504     movq    -48(%rbp), %rcx
505     movq    -56(%rbp), %rdx
506     movq    -8(%rbp), %rsi
507     movq    -40(%rbp), %rax
508     movl    %edi, -60(%rbp)
```

```
509     movss    -60(%rbp), %xmm0
510     movl    $0, %r8d
511     movq    %rax, %rdi
512     call    setup
513     movl    -12(%rbp), %edi
514     movq    -48(%rbp), %rcx
515     movq    -56(%rbp), %rdx
516     movq    -8(%rbp), %rsi
517     movq    -40(%rbp), %rax
518     movl    %edi, -60(%rbp)
519     movss    -60(%rbp), %xmm0
520     movl    $1, %r8d
521     movq    %rax, %rdi
522     call    setup
523     movl    -12(%rbp), %edi
524     movq    -48(%rbp), %rcx
525     movq    -56(%rbp), %rdx
526     movq    -8(%rbp), %rsi
527     movq    -40(%rbp), %rax
528     movl    %edi, -60(%rbp)
529     movss    -60(%rbp), %xmm0
530     movl    $2, %r8d
531     movq    %rax, %rdi
532     call    setup
533     movq    -8(%rbp), %rax
534     movl    $3, %esi
535     movq    %rax, %rdi
536     call    _free_matr
537     movl    $0, %eax
538 .L31:
539     leave
540     .cfi_def_cfa 7, 8
541     ret
542     .cfi_endproc
```

```
543 .LFE11:
544     .size    solve, .-solve
545     .section    .rodata
546 .LC8:
547     .string    "%f "
548     .text
549     .globl    write_data
550     .type    write_data, @function
551 write_data:
552 .LFB12:
553     .cfi_startproc
554     pushq    %rbp
555     .cfi_def_cfa_offset 16
556     .cfi_offset 6, -16
557     movq    %rsp, %rbp
558     .cfi_def_cfa_register 6
559     subq    $32, %rsp
560     movq    %rdi, -24(%rbp)
561     movl    $0, -4(%rbp)
562     jmp    .L34
563 .L35:
564     movl    -4(%rbp), %eax
565     cltq
566     leaq    0(,%rax,4), %rdx
567     movq    -24(%rbp), %rax
568     addq    %rdx, %rax
569     movss    (%rax), %xmm0
570     cvtss2sd    %xmm0, %xmm0
571     leaq    .LC8(%rip), %rdi
572     movl    $1, %eax
573     call    printf@PLT
574     addl    $1, -4(%rbp)
575 .L34:
576     cmpl    $2, -4(%rbp)
```

```
577     jle    .L35
578     movl    $10, %edi
579     call    putchar@PLT
580     nop
581     leave
582     .cfi_def_cfa 7, 8
583     ret
584     .cfi_endproc
585 .LFE12:
586     .size    write_data, .-write_data
587     .section    .rodata
588 .LC9:
589     .string    "More? "
590 .LC10:
591     .string    " %c"
592     .text
593     .globl    main
594     .type    main, @function
595 main:
596 .LFB13:
597     .cfi_startproc
598     pushq    %rbp
599     .cfi_def_cfa_offset 16
600     .cfi_offset 6, -16
601     movq    %rsp, %rbp
602     .cfi_def_cfa_register 6
603     subq    $48, %rsp
604     movq    %fs:40, %rax
605     movq    %rax, -8(%rbp)
606     xorl    %eax, %eax
607     movl    $3, %esi
608     movl    $3, %edi
609     call    _alloc_matr
610     movq    %rax, -32(%rbp)
```

```
611     movl    $12, %edi
612     call    malloc@PLT
613     movq    %rax, -24(%rbp)
614     movl    $12, %edi
615     call    malloc@PLT
616     movq    %rax, -16(%rbp)
617 .L40:
618     movq    -24(%rbp), %rdx
619     movq    -32(%rbp), %rax
620     movq    %rdx, %rsi
621     movq    %rax, %rdi
622     call    get_data
623     movq    -16(%rbp), %rdx
624     movq    -24(%rbp), %rcx
625     movq    -32(%rbp), %rax
626     movq    %rcx, %rsi
627     movq    %rax, %rdi
628     call    solve
629     movb    %al, -33(%rbp)
630     movzbl    -33(%rbp), %eax
631     xorl    $1, %eax
632     testb    %al, %al
633     je    .L37
634     movq    -16(%rbp), %rax
635     movq    %rax, %rdi
636     call    write_data
637 .L37:
638     leaq    .LC9(%rip), %rdi
639     movl    $0, %eax
640     call    printf@PLT
641     leaq    -34(%rbp), %rax
642     movq    %rax, %rsi
643     leaq    .LC10(%rip), %rdi
644     movl    $0, %eax
```

```
645     call    __isoc99_scanf@PLT
646     movzbl    -34(%rbp), %eax
647     cmpb    $121, %al
648     jne    .L44
649     jmp    .L40
650 .L44:
651     nop
652     movq    -24(%rbp), %rax
653     movq    %rax, %rdi
654     call    free@PLT
655     movq    -16(%rbp), %rax
656     movq    %rax, %rdi
657     call    free@PLT
658     movq    -32(%rbp), %rax
659     movl    $3, %esi
660     movq    %rax, %rdi
661     call    _free_matr
662     movl    $0, %eax
663     movq    -8(%rbp), %rcx
664     xorq    %fs:40, %rcx
665     je    .L42
666     call    __stack_chk_fail@PLT
667 .L42:
668     leave
669     .cfi_def_cfa 7, 8
670     ret
671     .cfi_endproc
672 .LFE13:
673     .size    main, .-main
674     .ident    "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
675     .section    .note.GNU-stack,"",@progbits
```

ПРИЛОЖЕНИЕ Г

Результаты ручного подсчёта для C

Таблица 12. Результаты ручного подсчёта для C

Оператор	Количество
*	9
+	1
-	6
/	1
;	18
<	3
=	9
>	1
++	3
[]	40
for	3
deter	3
setup	4
solve	2
return	3
и ()	12

(a) Операторы

Операнд	Количество
0	14
1	14
2	12
a	21
b	5
j	10
i	10
det	5
CMAX	2
RMAX	3
coef	5
true	1
false	1

(b) Операнды

ПРИЛОЖЕНИЕ Д

Результаты ручного подсчёта для Pascal

Таблица 13. Результаты ручного подсчёта для Pascal

Оператор	Количество
*	9
-	6
/	1
;	15
>	1
:=	12
<>	2
[N]	2
”[N N]”	20
if then	1
For .. do	3
if then else	1
function deter	3
procedure setup	4
procedure solve	2
begin end или ()	11

(a) Операторы

Операнд	Количество
1	15
2	12
3	12
a	20
b	5
n	4
y	3
0.0	1
sum	2
coef	5
true	1
error	3
false	1

(b) Операнды

ПРИЛОЖЕНИЕ Е

Результаты ручного подсчёта для ассемблера

Таблица 14. Результаты ручного подсчёта операторов для ассемблера

Оператор	Количество
0	27
-4	16
-8	25
-12	7
-16	6
-20	5
-24	5
-32	6
-36	1
-40	8
-48	6
-52	6
-56	6
-60	6
nop	3
ret	5
addq	35
cltq	8
cmpl	5
leaq	10
movb	1
movl	38
movq	88
popq	1
pxor	3
salq	5
subq	7
xorl	1

Таблица 14. Результаты ручного подсчёта операторов для ассемблера

Оператор	Количество
divss	1
leave	4
movss	28
mulss	9
pushq	7
subss	4
subss	4
movaps	1
je .L42	1
jp .L29	1
ucomiss	2
cvtss2sd	1
jle .L23	1
jle .L27	1
jle .L28	1
jle .L35	1
jmp .L21	1
jmp .L25	1
jmp .L26	1
jmp .L31	1
jmp .L34	1
jne .L29	1
call deter	2
call setup	3

Таблица 15. Результаты ручного подсчёта операндов для ассемблера

Оператор	Количество
8	5
$0	8

Таблица 15. Результаты ручного подсчёта операндов для ассемблера

Оператор	Количество
`$1`	14
`$2`	10
`$3`	4
`$4`	9
`$8`	11
`fs`	2
`$16`	6
`$40`	1
`$56`	11
`$64`	1
`eax`	16
`ecx`	6
`edi`	9
`edx`	4
`esi`	3
`r8d`	4
`rax`	123
`rbp`	111
`rbx`	1
`rcx`	44
`rdi`	13
`rdx`	26
`rsp`	11
`xmm0`	36
`xmm1`	12
`xmm2`	12
`xmm3`	6

ПРИЛОЖЕНИЕ Ж

Логи работы

```
Statistics for module c.lxm
=====================================
The number of different operators     : 35
The number of different operands      : 27
The total number of operators         : 300
The total number of operands          : 234

Dictionary                    (  D)    : 62
Length                        (  N)    : 534
Length estimation             ( ^N)    : 307.907
Volume                        (  V)    : 3179.54
Potential volume              ( *V)    : 11.6096
Limit volume                  (**V)    : 15.6844
Programming level             (  L)    : 0.00365136
Programming level estimation  ( ^L)    : 0.00659341
Intellect                     (  I)    : 20.964
Time of programming           (  T)    : 87078.3
Time estimation               ( ^T)    : 27805.6
Programming language level    (lambda) : 0.0423909
Work on programming           (  E)    : 870783
Error                         (  B)    : 3.03962
Error estimation              ( ^B)    : 1.05985


Table:
=====================================
Operators:
|  1    |  1    | !
|  2    |  1    | !=
|  3    |  31    | ()
|  4    |  13    | *
|  5    |  1    | +
```

```
33  |  6    |  10    | ++
34  |  7    |  42    | ,
35  |  8    |  6    | -
36  |  9    |  1    | /
37  |  10    |  10    | <
38  |  11    |  23    | =
39  |  12    |  1    | ==
40  |  13    |  1    | >
41  |  14    |  4    | SIZEOF
42  |  15    |  51    | []
43  |  16    |  3    | _&
44  |  17    |  37    | __*
45  |  18    |  3    | _alloc_matr
46  |  19    |  3    | _free_matr
47  |  20    |  1    | break
48  |  21    |  3    | deter
49  |  22    |  10    | for
50  |  23    |  4    | free
51  |  24    |  2    | get_data
52  |  25    |  4    | if
53  |  26    |  1    | main
54  |  27    |  4    | malloc
55  |  28    |  2    | print_matr
56  |  29    |  10    | printf
57  |  30    |  5    | return
58  |  31    |  3    | scanf
59  |  32    |  4    | setup
60  |  33    |  2    | solve
61  |  34    |  1    | while
62  |  35    |  2    | write_data
63  Operands:
64  |  1    |  1    | ” %c”
65  |  2    |  1    | ”%d: ”
66  |  3    |  1    | ”%f  ”
```

```
 67  |  4    |  1    | ”%f ”
 68  |  5    |  2    | ”%f”
 69  |  6    |  1    | ”: %f\n”
 70  |  7    |  1    | ”C: ”
 71  |  8    |  1    | ”ERROR: matrix is singular.”
 72  |  9    |  1    | ”Equation %d\n”
 73  |  10    |  1    | ”More? ”
 74  |  11    |  2    | ”\n”
 75  |  12    |  1    | 'y'
 76  |  13    |  24    | 0
 77  |  14    |  16    | 1
 78  |  15    |  12    | 2
 79  |  16    |  9    | CMAX
 80  |  17    |  8    | RMAX
 81  |  18    |  36    | a
 82  |  19    |  13    | b
 83  |  20    |  12    | coef
 84  |  21    |  8    | det
 85  |  22    |  3    | error
 86  |  23    |  35    | i
 87  |  24    |  20    | j
 88  |  25    |  6    | m
 89  |  26    |  3    | scan
 90  |  27    |  15    | y
 91
 92
 93  Summary:
 94  ====================================
 95  The number of different operators     : 35
 96  The number of different operands      : 27
 97  The total number of operators         : 300
 98  The total number of operands          : 234
 99
100  Dictionary                  (  D)     : 62
```

```
Length                        (  N)    : 534
Length estimation             ( ^N)    : 307.907
Volume                        (  V)    : 3179.54
Potential volume              ( *V)    : 11.6096
Limit volume                  (**V)    : 15.6844
Programming level             (  L)    : 0.00365136
Programming level estimation  ( ^L)    : 0.00659341
Intellect                     (  I)    : 20.964
Time of programming           (  T)    : 87078.3
Time estimation               ( ^T)    : 27805.6
Programming language level    (lambda) : 0.0423909
Work on programming           (  E)    : 870783
Error                         (  B)    : 3.03962
Error estimation              ( ^B)    : 1.05985

Statistics for module pas.lxm
=====================================
The number of different operators     : 29
The number of different operands      : 36
The total number of operators         : 134
The total number of operands          : 193

Dictionary                    (  D)    : 65
Length                        (  N)    : 327
Length estimation             ( ^N)    : 326.999
Volume                        (  V)    : 1969.31
Potential volume              ( *V)    : 11.6096
Limit volume                  (**V)    : 15.6844
Programming level             (  L)    : 0.00589527
Programming level estimation  ( ^L)    : 0.012864
Intellect                     (  I)    : 25.3333
Time of programming           (  T)    : 33405
Time estimation               ( ^T)    : 15308.6
Programming language level    (lambda) : 0.068442
Work on programming           (  E)    : 334050
```

```
Error                          (  B)    : 1.6048
Error estimation               ( ^B)    : 0.656438


Table:
====================================
Operators:
|  1    |  20    | ()
|  2    |  9    | *
|  3    |  1    | +
|  4    |  6    | -
|  5    |  1    | /
|  6    |  2    | <>
|  7    |  15    | =
|  8    |  1    | >
|  9    |  2    | ClrScr
|  10    |  27    | []
|  11    |  1    | and
|  12    |  1    | chr
|  13    |  1    | const
|  14    |  3    | deter
|  15    |  8    | for
|  16    |  2    | get_data
|  17    |  3    | if
|  18    |  1    | not
|  19    |  1    | program
|  20    |  1    | read
|  21    |  2    | readln
|  22    |  1    | real
|  23    |  1    | repeat
|  24    |  4    | setup
|  25    |  2    | solve
|  26    |  1    | type
|  27    |  5    | write
```

```
|  28    |  2    | write_data
|  29    |  10    | writeln
Operands:
|  1    |  1    | ' '
|  2    |  1    | ' Equation'
|  3    |  1    | ',C:'
|  4    |  2    | ':'
|  5    |  1    | 'ERROR: matrix is singular.'
|  6    |  1    | 'More?'
|  7    |  1    | 'Simultaneous solution by Cramers rule'
|  8    |  1    | 'Y'
|  9    |  1    | 'y'
|  10    |  1    | 0.0
|  11    |  23    | 1
|  12    |  12    | 2
|  13    |  16    | 3
|  14    |  2    | 4
|  15    |  1    | 5
|  16    |  3    | 7
|  17    |  1    | 9
|  18    |  25    | a
|  19    |  1    | ary2s
|  20    |  1    | arys
|  21    |  10    | b
|  22    |  3    | cmax
|  23    |  9    | coef
|  24    |  4    | det
|  25    |  1    | deter
|  26    |  6    | error
|  27    |  1    | false
|  28    |  16    | i
|  29    |  13    | j
|  30    |  14    | n
|  31    |  3    | rmax
```

```
 89  |  32    |  1    | simq1
 90  |  33    |  3    | sum
 91  |  34    |  1    | true
 92  |  35    |  8    | y
 93  |  36    |  4    | yesno
 94
 95
 96  Summary:
 97  ====================================
 98  The number of different operators     : 29
 99  The number of different operands      : 36
100  The total number of operators         : 134
101  The total number of operands          : 193
102
103  Dictionary                  (  D)    : 65
104  Length                      (  N)    : 327
105  Length estimation           ( ^N)    : 326.999
106  Volume                      (  V)    : 1969.31
107  Potential volume            ( *V)    : 11.6096
108  Limit volume                (**V)    : 15.6844
109  Programming level           (  L)    : 0.00589527
110  Programming level estimation ( ^L)   : 0.012864
111  Intellect                   (  I)    : 25.3333
112  Time of programming         (  T)    : 33405
113  Time estimation             ( ^T)    : 15308.6
114  Programming language level  (lambda) : 0.068442
115  Work on programming         (  E)    : 334050
116  Error                       (  B)    : 1.6048
117  Error estimation            ( ^B)    : 0.656438
```