

**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. 7304

Моторин Е.В

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

## **Цель работы**

Измерить и сравнить метрики Холстеда для программ для C, Pascal и ас-семблере.

## **Задание**

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.

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

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.

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.

## Ход работы:

Вариант — 11, “Решение системы уравнений методом Крамера”

## 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.

Таблица 1. Измеримые характеристики для языка Pascal

Описание	Значение
Число простых (уникальных) операторов	16
Число простых (уникальных) операндов	13
Общее число всех операторов	93
Общее число всех операндов	84
Словарь	29
Опытная длина	177

Таблица 2. Расчётные характеристики для языка Pascal

Описание	Значение
Теоретическая длина	112.11
Объем	859.86
Потенциальный объем	11.61
Уровень	0.013502
Интеллектуальное содержание	16.63
Работа по программированию	63685.33
Время программирования	6368.53
Уровень языка	0.16
Количество ошибок	1

Таблица 3. Измеримые характеристики для языка C

Описание	Значение
Число простых (уникальных) операторов	16
Число простых (уникальных) операндов	13
Общее число всех операторов	118
Общее число всех операндов	103
Словарь	29
Опытная длина	221

Таблица 4. Расчётные характеристики для языка C

Описание	Значение
Теоретическая длина	112.11
Объем	1073.61
Потенциальный объем	11.61
Уровень	0.010814
Интеллектуальное содержание	16.94
Работа по программированию	99283.57
Время программирования	9928.36
Уровень языка	0.13
Количество ошибок	2

Таблица 5. Измеримые характеристики для языка ассемблер

Описание	Значение
Число простых (уникальных) операторов	52
Число простых (уникальных) операндов	29
Общее число всех операторов	418
Общее число всех операндов	519
Словарь	81
Опытная длина	937

Таблица 6. Расчётные характеристики для языка ассемблер

<b>Описание</b>	<b>Значение</b>
Теоретическая длина	437.30
Объем	5940.44
Потенциальный объем	11.61
Уровень	0.001954
Интеллектуальное содержание	12.77
Работа по программированию	3039613.58
Время программирования	303961.36
Уровень языка	0.02
Количество ошибок	6

## 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

Описание	Значение
Число простых (уникальных) операторов	29
Число простых (уникальных) операндов	36
Общее число всех операторов	134
Общее число всех операндов	193
Словарь	65
Опытная длина	327

Таблица 8. Расчётные характеристики для языка Pascal

Описание	Значение
Теоретическая длина	327.00
Объем	1969.31
Потенциальный объем	11.61
Уровень	0.005895
Интеллектуальное содержание	25.33
Работа по программированию	334050
Время программирования	33405
Уровень языка	0.07
Количество ошибок	2

Таблица 9. Измеримые характеристики для языка С

Описание	Значение
Число простых (уникальных) операторов	16
Число простых (уникальных) операндов	13
Общее число всех операторов	118
Общее число всех операндов	103
Словарь	29
Опытная длина	221

Таблица 10. Расчётные характеристики для языка С

Описание	Значение
Теоретическая длина	307
Объем	3179.54
Потенциальный объем	11.61
Уровень	0.003651
Интеллектуальное содержание	20.96
Работа по программированию	870783
Время программирования	87078.30
Уровень языка	0.04
Количество ошибок	4

Таблица 11. Сводная таблица

Описание	Pascal (ручн.)	C (ручн.)	Asm (ручн.)	Pascal (прог.)	C (прог.)
Число простых (уникальных) операторов	16	16	52	29	35
Число простых (уникальных) операндов	13	13	29	36	27
Общее число всех операторов	93	118	418	134	300
Общее число всех операндов	84	103	519	194	234
Словарь	29	29	81	65	62
Экспериментальная длина	177	221	937	327	534
Теоретическая длина	112.11	112.11	473.3	372	307
Объем	859.86	1073.61	5940.44	1969.31	3179.54
Потенциальный объем	11.61	11.61	11.61	11.61	11.61
Уровень	0.013502	0.010814	0.001954	0.005895	0.003651
Интеллектуальное содержание	16.63	16.94	12.77	25.33	20.96
Работа по программированию	63685.33	99283.57	3039613.58	334050	870783
Время программирования	6368.53	9928.36	303961.36	33405	87078.30
Уровень языка	0.16	0.13	0.02	0.07	0.04
Количество ошибок	1	2	6	2	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 > Ас-семблер. 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

```
program simq1;
{ pascal program to solve three simultaneous equations by Cramer's rule }

const rmax = 3;
      cmax = 3;

type  arys  = array[1..cmax] of real;
      ary2s = array[1..rmax,1..cmax] of real;

var   y,coef: arys;
      a      : ary2s;
      n      : integer;
      yesno : char;
      error  : boolean;

procedure get_data(var a: ary2s;
                  var y: arys;
                  var n: integer);

{ get the values for n, and arrays a,y }

var   i,j      : integer;

begin { procedure get_data }
  writeln;
  n:=rmax;
  for i:=1 to n do
    begin
      writeln(' Equation',i:3);
      for j:=1 to n do
        begin
          write(j:3,':');
          read(a[i,j])
        end;
      write(',C:');
      readln(y[i])
    end;
  writeln;
```

```

for i:=1 to n do
begin
  for j:=1 to n do
    write(a[i,j]:7:4,' ');
    writeln(':',y[i]:7:4)
  end;
  writeln
end;      { procedure get_data }

procedure write_data;
  { print out the answeres }

var   i      : integer;

begin { write_data }
  for i:=1 to n do
    write(coef[i]:9:5);
  writeln
end;      { write_data }

procedure solve(a: ary2s; y: arys;
  var coef: arys;    n: integer;
  var error: boolean);

var
  b      : ary2s;
  i,j    : integer;
  det    : real;

function deter(a: ary2s): real;
{ pascal program to calculate the determinant of a 3-by-3matrix }

var
  sum    : real;

begin { function deter }
  sum:=a[1,1]*(a[2,2]*a[3,3]-a[3,2]*a[2,3])
    -a[1,2]*(a[2,1]*a[3,3]-a[3,1]*a[2,3])
    +a[1,3]*(a[2,1]*a[3,2]-a[3,1]*a[2,2]);
  deter:=sum
end;  { function deter }

```

```

procedure setup(var b: ary2s;
                var coef: arys;
                j: integer);

var    i        : integer;

begin { setup }
  for i:=1 to n do
    begin
      b[i,j]:=y[i];
      if j>1 then b[i,j-1]:=a[i,j-1]
    end;
    coef[j]:=deter(b)/det
end; { setup }

begin          { procedure solve }
  error:=false;
  for i:=1 to n do
    for j:=1 to n do
      b[i,j]:=a[i,j];
    det:=deter(b);
    if det=0.0 then
      begin
        error:=true;
        writeln(chr(7),'ERROR: matrix is singular.')
      end
    else
      begin
        setup(b,coef,1);
        setup(b,coef,2);
        setup(b,coef,3);
      end { else }
    end; {procedure solve }

begin          { MAIN program }
  ClrScr;
  writeln;
  writeln('Simultaneous solution by Cramers rule');
  repeat
    get_data(a,y,n);

```

```
solve(a,y,coef,n,error);  
if not error then write_data;  
writeln;  
write('More?');  
readln(yesno);  
ClrScr  
until(yesno<>'Y')and(yesno<>'y')  
end.
```

## ПРИЛОЖЕНИЕ Б

### Код программы на С

```
#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]);
}
}

/* get the values for n, and arrays a,y */
void get_data(float** a, float* y) {
for (int i = 0; i < RMAX; i++) {
printf("Equation %d\n", i);
for (int j = 0; j < CMAX; j++) {
printf("%d: ", j);
scanf("%f", &a[i][j]);
}
}
```

```

printf("C: ");
scanf("%f", &y[i]);
}
print_matr(a, y);
printf("\n");
}

```

```

/* pascal program to calculate the determinant of a 3-by-3matrix */
float deter(float** a) {
return a[0][0] * (a[1][1] * a[2][2] - a [2][1] * a[1][2])
- a[0][1] * (a[1][0] * a[2][2] - a [2][0] * a[1][2])
+ a[0][2] * (a[1][0] * a[2][1] - a [2][0] * a[1][1]);
}

```

```

void setup(float** a, float** b, float* coef, float* y, int j, float det) {
for (int i = 0; i < RMAX; i++) {
b[i][j] = y[i];
if (j > 0) {
b[i][j-1] = a[i][j-1];
}
}
coef[j] = deter(b) / det;
}

```

```

int solve(float** a, float* y, float* coef) {
float** b = _alloc_matr(RMAX, CMAX);
float det = 0;
for (int i = 0; i < RMAX; i++) {
for (int j = 0; j < CMAX; j++) {
b[i][j] = a[i][j];
}
}
det = deter(b);
if (det == 0) {
printf("ERROR: matrix is singular.");
return 1;
}
setup(a, b, coef, y, 0, det);
setup(a, b, coef, y, 1, det);
setup(a, b, coef, y, 2, det);
_free_matr(b, RMAX);
return 0;
}

```

```
}
```

```
void write_data(float* coef) {  
    for (int i = 0; i < CMAX; i++) {  
        printf("%f ", coef[i]);  
    }  
    printf("\n");  
}
```

```
int main() {  
    float** a = _alloc_matr(RMAX, CMAX);  
    float* y = (float*)malloc(CMAX * sizeof(float));  
    float* coef = (float*)malloc(CMAX * sizeof(float));  
    int error;  
    char scan;  
    while (1) {  
        get_data(a, y);  
        error = solve(a, y, coef);  
        if (!error) {  
            write_data(coef);  
        }  
        printf("More? ");  
        scanf(" %c", &scan);  
        if (scan != 'y') {  
            break;  
        }  
    }  
    free(y);  
    free(coef);  
    _free_matr(a, RMAX);  
    return 0;  
}
```

## ПРИЛОЖЕНИЕ В

### Код программы на ассемблере

```
_alloc_matr:
push rbp
mov rbp, rsp
push rbx
sub rsp, 40
mov DWORD PTR [rbp-36], edi
mov DWORD PTR [rbp-40], esi
mov eax, DWORD PTR [rbp-36]
cdqe
sal rax, 3
mov rdi, rax
call malloc
mov QWORD PTR [rbp-32], rax
mov DWORD PTR [rbp-20], 0
jmp .L2
.L3:
mov eax, DWORD PTR [rbp-40]
cdqe
sal rax, 2
mov edx, DWORD PTR [rbp-20]
movsx rdx, edx
lea rcx, [0+rdx*8]
mov rdx, QWORD PTR [rbp-32]
lea rbx, [rcx+rdx]
mov rdi, rax
call malloc
mov QWORD PTR [rbx], rax
add DWORD PTR [rbp-20], 1
.L2:
cmp DWORD PTR [rbp-20], 2
jle .L3
mov rax, QWORD PTR [rbp-32]
mov rbx, QWORD PTR [rbp-8]
leave
ret
_free_matr:
push rbp
mov rbp, rsp
sub rsp, 32
mov QWORD PTR [rbp-24], rdi
```



```

mov DWORD PTR [rbp-28], esi
mov DWORD PTR [rbp-4], 0
jmp .L6
.L7:
mov eax, DWORD PTR [rbp-4]
cdqe
lea rdx, [0+rax*8]
mov rax, QWORD PTR [rbp-24]
add rax, rdx
mov rax, QWORD PTR [rax]
mov rdi, rax
call free
add DWORD PTR [rbp-4], 1
.L6:
mov eax, DWORD PTR [rbp-4]
cmp eax, DWORD PTR [rbp-28]
jl .L7
mov rax, QWORD PTR [rbp-24]
mov rdi, rax
call free
nop
leave
ret
.LC0:
.string "%f "
.LC1:
.string ": %f\n"
print_matr:
push rbp
mov rbp, rsp
sub rsp, 32
mov QWORD PTR [rbp-24], rdi
mov QWORD PTR [rbp-32], rsi
mov DWORD PTR [rbp-4], 0
jmp .L9
.L12:
mov DWORD PTR [rbp-8], 0
jmp .L10
.L11:
mov eax, DWORD PTR [rbp-4]
cdqe
lea rdx, [0+rax*8]

```

```

mov rax, QWORD PTR [rbp-24]
add rax, rdx
mov rdx, QWORD PTR [rax]
mov eax, DWORD PTR [rbp-8]
cdqe
sal rax, 2
add rax, rdx
movss xmm0, DWORD PTR [rax]
pxor xmm1, xmm1
cvtss2sd xmm1, xmm0
movq rax, xmm1
movq xmm0, rax
mov edi, OFFSET FLAT:.LC0
mov eax, 1
call printf
add DWORD PTR [rbp-8], 1
.L10:
cmp DWORD PTR [rbp-8], 2
jle .L11
mov eax, DWORD PTR [rbp-4]
cdqe
lea rdx, [0+rax*4]
mov rax, QWORD PTR [rbp-32]
add rax, rdx
movss xmm0, DWORD PTR [rax]
pxor xmm2, xmm2
cvtss2sd xmm2, xmm0
movq rax, xmm2
movq xmm0, rax
mov edi, OFFSET FLAT:.LC1
mov eax, 1
call printf
add DWORD PTR [rbp-4], 1
.L9:
cmp DWORD PTR [rbp-4], 2
jle .L12
nop
nop
leave
ret
.LC2:
.string "Equation %d\n"

```

```
.LC3:
.string "%d: "
.LC4:
.string "%f"
.LC5:
.string "C: "
get_data:
push rbp
mov rbp, rsp
sub rsp, 32
mov QWORD PTR [rbp-24], rdi
mov QWORD PTR [rbp-32], rsi
mov DWORD PTR [rbp-4], 0
jmp .L14
.L17:
mov eax, DWORD PTR [rbp-4]
mov esi, eax
mov edi, OFFSET FLAT:.LC2
mov eax, 0
call printf
mov DWORD PTR [rbp-8], 0
jmp .L15
.L16:
mov eax, DWORD PTR [rbp-8]
mov esi, eax
mov edi, OFFSET FLAT:.LC3
mov eax, 0
call printf
mov eax, DWORD PTR [rbp-4]
cdqe
lea rdx, [0+rax*8]
mov rax, QWORD PTR [rbp-24]
add rax, rdx
mov rdx, QWORD PTR [rax]
mov eax, DWORD PTR [rbp-8]
cdqe
sal rax, 2
add rax, rdx
mov rsi, rax
mov edi, OFFSET FLAT:.LC4
mov eax, 0
call scanf
```

```

add DWORD PTR [rbp-8], 1
.L15:
cmp DWORD PTR [rbp-8], 2
jle .L16
mov edi, OFFSET FLAT:.LC5
mov eax, 0
call printf
mov eax, DWORD PTR [rbp-4]
cdqe
lea rdx, [0+rax*4]
mov rax, QWORD PTR [rbp-32]
add rax, rdx
mov rsi, rax
mov edi, OFFSET FLAT:.LC4
mov eax, 0
call scanf
add DWORD PTR [rbp-4], 1
.L14:
cmp DWORD PTR [rbp-4], 2
jle .L17
mov rdx, QWORD PTR [rbp-32]
mov rax, QWORD PTR [rbp-24]
mov rsi, rdx
mov rdi, rax
call print_matr
mov edi, 10
call putchar
nop
leave
ret
deter:
push rbp
mov rbp, rsp
mov QWORD PTR [rbp-8], rdi
mov rax, QWORD PTR [rbp-8]
mov rax, QWORD PTR [rax]
movss xmm1, DWORD PTR [rax]
mov rax, QWORD PTR [rbp-8]
add rax, 8
mov rax, QWORD PTR [rax]
add rax, 4
movss xmm2, DWORD PTR [rax]

```

```
mov rax, QWORD PTR [rbp-8]
add rax, 16
mov rax, QWORD PTR [rax]
add rax, 8
movss xmm0, DWORD PTR [rax]
mulss xmm0, xmm2
mov rax, QWORD PTR [rbp-8]
add rax, 16
mov rax, QWORD PTR [rax]
add rax, 4
movss xmm3, DWORD PTR [rax]
mov rax, QWORD PTR [rbp-8]
add rax, 8
mov rax, QWORD PTR [rax]
add rax, 8
movss xmm2, DWORD PTR [rax]
mulss xmm2, xmm3
subss xmm0, xmm2
mulss xmm0, xmm1
mov rax, QWORD PTR [rbp-8]
mov rax, QWORD PTR [rax]
add rax, 4
movss xmm2, DWORD PTR [rax]
mov rax, QWORD PTR [rbp-8]
add rax, 8
mov rax, QWORD PTR [rax]
movss xmm3, DWORD PTR [rax]
mov rax, QWORD PTR [rbp-8]
add rax, 16
mov rax, QWORD PTR [rax]
add rax, 8
movss xmm1, DWORD PTR [rax]
mulss xmm1, xmm3
mov rax, QWORD PTR [rbp-8]
add rax, 16
mov rax, QWORD PTR [rax]
movss xmm4, DWORD PTR [rax]
mov rax, QWORD PTR [rbp-8]
add rax, 8
mov rax, QWORD PTR [rax]
add rax, 8
movss xmm3, DWORD PTR [rax]
```

```
mulss xmm3, xmm4
subss xmm1, xmm3
mulss xmm2, xmm1
movaps xmm1, xmm0
subss xmm1, xmm2
mov rax, QWORD PTR [rbp-8]
mov rax, QWORD PTR [rax]
add rax, 8
movss xmm2, DWORD PTR [rax]
mov rax, QWORD PTR [rbp-8]
add rax, 8
mov rax, QWORD PTR [rax]
movss xmm3, DWORD PTR [rax]
mov rax, QWORD PTR [rbp-8]
add rax, 16
mov rax, QWORD PTR [rax]
add rax, 4
movss xmm0, DWORD PTR [rax]
mulss xmm0, xmm3
mov rax, QWORD PTR [rbp-8]
add rax, 16
mov rax, QWORD PTR [rax]
movss xmm4, DWORD PTR [rax]
mov rax, QWORD PTR [rbp-8]
add rax, 8
mov rax, QWORD PTR [rax]
add rax, 4
movss xmm3, DWORD PTR [rax]
mulss xmm3, xmm4
subss xmm0, xmm3
mulss xmm0, xmm2
addss xmm0, xmm1
pop rbp
ret
setup:
push rbp
mov rbp, rsp
sub rsp, 56
mov QWORD PTR [rbp-24], rdi
mov QWORD PTR [rbp-32], rsi
mov QWORD PTR [rbp-40], rdx
mov QWORD PTR [rbp-48], rcx
```

```
mov DWORD PTR [rbp-52], r8d
movss DWORD PTR [rbp-56], xmm0
mov DWORD PTR [rbp-4], 0
jmp .L21
.L23:
mov eax, DWORD PTR [rbp-4]
cdqe
lea rdx, [0+rax*4]
mov rax, QWORD PTR [rbp-48]
add rdx, rax
mov eax, DWORD PTR [rbp-4]
cdqe
lea rcx, [0+rax*8]
mov rax, QWORD PTR [rbp-32]
add rax, rcx
mov rcx, QWORD PTR [rax]
mov eax, DWORD PTR [rbp-52]
cdqe
sal rax, 2
add rax, rcx
movss xmm0, DWORD PTR [rdx]
movss DWORD PTR [rax], xmm0
cmp DWORD PTR [rbp-52], 0
jle .L22
mov eax, DWORD PTR [rbp-4]
cdqe
lea rdx, [0+rax*8]
mov rax, QWORD PTR [rbp-24]
add rax, rdx
mov rdx, QWORD PTR [rax]
mov eax, DWORD PTR [rbp-52]
cdqe
sal rax, 2
sub rax, 4
add rdx, rax
mov eax, DWORD PTR [rbp-4]
cdqe
lea rcx, [0+rax*8]
mov rax, QWORD PTR [rbp-32]
add rax, rcx
mov rcx, QWORD PTR [rax]
mov eax, DWORD PTR [rbp-52]
```

```

cdqe
sal rax, 2
sub rax, 4
add rax, rcx
movss xmm0, DWORD PTR [rdx]
movss DWORD PTR [rax], xmm0
.L22:
add DWORD PTR [rbp-4], 1
.L21:
cmp DWORD PTR [rbp-4], 2
jle .L23
mov rax, QWORD PTR [rbp-32]
mov rdi, rax
call deter
movd eax, xmm0
mov edx, DWORD PTR [rbp-52]
movsx rdx, edx
lea rcx, [0+rdx*4]
mov rdx, QWORD PTR [rbp-40]
add rdx, rcx
movd xmm0, eax
divss xmm0, DWORD PTR [rbp-56]
movss DWORD PTR [rdx], xmm0
nop
leave
ret
.LC7:
.string "ERROR: matrix is singular."
solve:
push rbp
mov rbp, rsp
sub rsp, 64
mov QWORD PTR [rbp-40], rdi
mov QWORD PTR [rbp-48], rsi
mov QWORD PTR [rbp-56], rdx
mov esi, 3
mov edi, 3
call _alloc_matr
mov QWORD PTR [rbp-16], rax
pxor xmm0, xmm0
movss DWORD PTR [rbp-20], xmm0
mov DWORD PTR [rbp-4], 0

```



```

jmp .L25
.L28:
mov DWORD PTR [rbp-8], 0
jmp .L26
.L27:
mov eax, DWORD PTR [rbp-4]
cdqe
lea rdx, [0+rax*8]
mov rax, QWORD PTR [rbp-40]
add rax, rdx
mov rdx, QWORD PTR [rax]
mov eax, DWORD PTR [rbp-8]
cdqe
sal rax, 2
add rdx, rax
mov eax, DWORD PTR [rbp-4]
cdqe
lea rcx, [0+rax*8]
mov rax, QWORD PTR [rbp-16]
add rax, rcx
mov rcx, QWORD PTR [rax]
mov eax, DWORD PTR [rbp-8]
cdqe
sal rax, 2
add rax, rcx
movss xmm0, DWORD PTR [rdx]
movss DWORD PTR [rax], xmm0
add DWORD PTR [rbp-8], 1
.L26:
cmp DWORD PTR [rbp-8], 2
jle .L27
add DWORD PTR [rbp-4], 1
.L25:
cmp DWORD PTR [rbp-4], 2
jle .L28
mov rax, QWORD PTR [rbp-16]
mov rdi, rax
call deter
movd eax, xmm0
mov DWORD PTR [rbp-20], eax
pxor xmm0, xmm0
ucomiss xmm0, DWORD PTR [rbp-20]

```

```
jp .L29
pxor xmm0, xmm0
ucomiss xmm0, DWORD PTR [rbp-20]
jne .L29
mov edi, OFFSET FLAT:.LC7
mov eax, 0
call printf
mov eax, 1
jmp .L31
.L29:
mov edi, DWORD PTR [rbp-20]
mov rcx, QWORD PTR [rbp-48]
mov rdx, QWORD PTR [rbp-56]
mov rsi, QWORD PTR [rbp-16]
mov rax, QWORD PTR [rbp-40]
movd xmm0, edi
mov r8d, 0
mov rdi, rax
call setup
mov edi, DWORD PTR [rbp-20]
mov rcx, QWORD PTR [rbp-48]
mov rdx, QWORD PTR [rbp-56]
mov rsi, QWORD PTR [rbp-16]
mov rax, QWORD PTR [rbp-40]
movd xmm0, edi
mov r8d, 1
mov rdi, rax
call setup
mov edi, DWORD PTR [rbp-20]
mov rcx, QWORD PTR [rbp-48]
mov rdx, QWORD PTR [rbp-56]
mov rsi, QWORD PTR [rbp-16]
mov rax, QWORD PTR [rbp-40]
movd xmm0, edi
mov r8d, 2
mov rdi, rax
call setup
mov rax, QWORD PTR [rbp-16]
mov esi, 3
mov rdi, rax
call _free_matr
mov eax, 0
```

```

.L31:
leave
ret
.LC8:
.string "%f "
write_data:
push rbp
mov rbp, rsp
sub rsp, 32
mov QWORD PTR [rbp-24], rdi
mov DWORD PTR [rbp-4], 0
jmp .L34
.L35:
mov eax, DWORD PTR [rbp-4]
cdqe
lea rdx, [0+rax*4]
mov rax, QWORD PTR [rbp-24]
add rax, rdx
movss xmm0, DWORD PTR [rax]
pxor xmm1, xmm1
cvtss2sd xmm1, xmm0
movq rax, xmm1
movq xmm0, rax
mov edi, OFFSET FLAT:.LC8
mov eax, 1
call printf
add DWORD PTR [rbp-4], 1
.L34:
cmp DWORD PTR [rbp-4], 2
jle .L35
mov edi, 10
call putchar
nop
leave
ret
.LC9:
.string "More? "
.LC10:
.string " %c"
main:
push rbp
mov rbp, rsp

```

```
sub rsp, 32
mov esi, 3
mov edi, 3
call _alloc_matr
mov QWORD PTR [rbp-8], rax
mov edi, 12
call malloc
mov QWORD PTR [rbp-16], rax
mov edi, 12
call malloc
mov QWORD PTR [rbp-24], rax
.L40:
mov rdx, QWORD PTR [rbp-16]
mov rax, QWORD PTR [rbp-8]
mov rsi, rdx
mov rdi, rax
call get_data
mov rdx, QWORD PTR [rbp-24]
mov rcx, QWORD PTR [rbp-16]
mov rax, QWORD PTR [rbp-8]
mov rsi, rcx
mov rdi, rax
call solve
mov DWORD PTR [rbp-28], eax
cmp DWORD PTR [rbp-28], 0
jne .L37
mov rax, QWORD PTR [rbp-24]
mov rdi, rax
call write_data
.L37:
mov edi, OFFSET FLAT:.LC9
mov eax, 0
call printf
lea rax, [rbp-29]
mov rsi, rax
mov edi, OFFSET FLAT:.LC10
mov eax, 0
call scanf
movzx eax, BYTE PTR [rbp-29]
cmp al, 121
jne .L43
jmp .L40
```

```
.L43:
nop
mov rax, QWORD PTR [rbp-16]
mov rdi, rax
call free
mov rax, QWORD PTR [rbp-24]
mov rdi, rax
call free
mov rax, QWORD PTR [rbp-8]
mov esi, 3
mov rdi, rax
call _free_matr
mov eax, 0
leave
ret
```

## ПРИЛОЖЕНИЕ Г

### Результаты ручного подсчёта для С

Таблица 12. Результаты ручного подсчёта для С

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

## 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
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

## 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



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

8	5
\$0	8
\$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

1 Statistics for module c.lxm

2 =====

3 The number of different operators : 35

4 The number of different operands : 27

5 The total number of operators : 300

6 The total number of operands : 234

7

8 Dictionary ( D) : 62

9 Length ( N) : 534

10 Length estimation ( ^N) : 307.907

11 Volume ( V) : 3179.54

12 Potential volume ( \*V) : 11.6096

13 Limit volume (\*\*V) : 15.6844

14 Programming level ( L) : 0.00365136

15 Programming level estimation ( ^L) : 0.00659341

16 Intellect ( I) : 20.964

17 Time of programming ( T) : 87078.3

18 Time estimation ( ^T) : 27805.6

19 Programming language level (lambda) : 0.0423909

20 Work on programming ( E) : 870783

21 Error ( B) : 3.03962

22 Error estimation ( ^B) : 1.05985

23

24

25 Table:

26 =====

27 Operators:

28 | 1 | 1 | !

29 | 2 | 1 | !=

30 | 3 | 31 | ()

31 | 4 | 13 | \*

32 | 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

101	Length	( N)	: 534
102	Length estimation	( ^N)	: 307.907
103	Volume	( V)	: 3179.54
104	Potential volume	( *V)	: 11.6096
105	Limit volume	(**V)	: 15.6844
106	Programming level	( L)	: 0.00365136
107	Programming level estimation	( ^L)	: 0.00659341
108	Intellect	( I)	: 20.964
109	Time of programming	( T)	: 87078.3
110	Time estimation	( ^T)	: 27805.6
111	Programming language level	(lambda)	: 0.0423909
112	Work on programming	( E)	: 870783
113	Error	( B)	: 3.03962
114	Error estimation	( ^B)	: 1.05985

## 1 Statistics for module pas.lxm

2 =====

3	The number of different operators		: 29
4	The number of different operands		: 36
5	The total number of operators		: 134
6	The total number of operands		: 193
7			
8	Dictionary	( D)	: 65
9	Length	( N)	: 327
10	Length estimation	( ^N)	: 326.999
11	Volume	( V)	: 1969.31
12	Potential volume	( *V)	: 11.6096
13	Limit volume	(**V)	: 15.6844
14	Programming level	( L)	: 0.00589527
15	Programming level estimation	( ^L)	: 0.012864
16	Intellect	( I)	: 25.3333
17	Time of programming	( T)	: 33405
18	Time estimation	( ^T)	: 15308.6
19	Programming language level	(lambda)	: 0.068442
20	Work on programming	( E)	: 334050

21 Error ( B) : 1.6048  
 22 Error estimation ( ^B) : 0.656438  
 23  
 24

25 Table:

26 =====

27 Operators:

28	1	20	()
29	2	9	*
30	3	1	+
31	4	6	-
32	5	1	/
33	6	2	<>
34	7	15	=
35	8	1	>
36	9	2	ClrScr
37	10	27	[]
38	11	1	and
39	12	1	chr
40	13	1	const
41	14	3	deter
42	15	8	for
43	16	2	get_data
44	17	3	if
45	18	1	not
46	19	1	program
47	20	1	read
48	21	2	readln
49	22	1	real
50	23	1	repeat
51	24	4	setup
52	25	2	solve
53	26	1	type
54	27	5	write

55		28		2		write_data
56		29		10		writeln
57	Operands:					
58		1		1		''
59		2		1		' Equation'
60		3		1		','C:'
61		4		2		':'
62		5		1		'ERROR: matrix is singular.'
63		6		1		'More?'
64		7		1		'Simultaneous solution by Cramers rule'
65		8		1		'Y'
66		9		1		'y'
67		10		1		0.0
68		11		23		1
69		12		12		2
70		13		16		3
71		14		2		4
72		15		1		5
73		16		3		7
74		17		1		9
75		18		25		a
76		19		1		ary2s
77		20		1		arys
78		21		10		b
79		22		3		cmax
80		23		9		coef
81		24		4		det
82		25		1		deter
83		26		6		error
84		27		1		false
85		28		16		i
86		29		13		j
87		30		14		n
88		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