

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

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

Студент гр. 6304

Пискунов Я.А.

Преподаватель

Кирияничков В.А.

Санкт-Петербург

2020

Цель работы.

Изучение способа расчета метрических характеристик качества разработки программ на основе метрик Холстеда.

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. Добиться, чтобы программы на Паскале и Си были работоспособны и давали корректные результаты (это потребуется в дальнейшем при проведении с ними измерительных экспериментов).

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.

Ход работы.

Для начала определим метрические характеристики у исходного файла на Pascal. Перед проведением измерением были исправлены ошибки (в некоторых местах отсутствовали точки с запятой). Результаты ручного измерения характеристик представлены в табл. 1.

Таблица 1 – Ручной расчёт характеристик программы на Pascal

№	Оператор	Количество	№	Операнд	Количество
1	;	27	1	y	5
2	:=	18	2	coef	6
3	=	1	3	a	20
4	() или begin end	12	4	n	11
5	if ... then ...	3	5	yesno	3
6	for ... to ... do ...	8	6	error	4
7	repeat ... until ...	1	7	i	17
8	[]	22	8	j	13
9	+	1	9	b	7
10	-	6	10	det	3
11	/	1	11	sum	2
12	*	9	12	deter	1
13	>	1	13	true	1
14	<>	2	14	false	1
15	not	1	15	3	12
16	get_data	1	16	1	20
17	write_data	1	17	2	12
18	solve	1	18	0.0	1
19	setup	3	19	rmax	1
20	deter	2			
Всего		131	Всего		140

Далее произведем автоматический расчет метрик. Его результаты представлены в табл. 2. Из результатов также убраны операторы, которые обеспечивают интерфейс с пользователем.

Таблица 2 – Автоматический расчёт характеристик программы наPascal

№	Оператор	Количество	№	Операнд	Количество
1	()	20	1	0.0	1
2	*	9	2	1	23
3	+	1	3	2	12
4	-	6	4	3	16
5	/	1	5	4	2
6	;	117	6	5	1
7	◊	2	7	7	3
8	=	15	8	9	1
9	>	1	9	a	25
10	deter	3	10	ary2s	1
11	[]	27	11	arys	1
12	and	1	12	b	10
13	for	8	13	cmax	3
14	get_data	2	14	coef	9
15	if	3	15	det	4
16	not	1	16	deter	1
17	repeat	1	17	error	6
18	stup	4	18	false	1
19	solve	2	19	i	16
20	write_data	2	20	j	13
21	type	1	21	n	14
22	real	1	22	rmax	3
23	Program	1	23	simq1	1
24	const	1	24	sum	3
25	chr	1	25	true	1
			26	y	8
			27	yesno	4
Всего		233	Всего		183

Результаты расчетов метрических характеристик представлены в табл. 6.

Далее произведем аналогичный подсчет для программы на языке С.

Результаты ручного подсчет представлены в табл. 3, а программного – в табл.4.

Таблица 3 – Ручной расчёт характеристик программы на С

№	Оператор	Количество	№	Операнд	Количество
1	;	40	1	CMAX	4
2	=	24	2	RMAX	5
3	() или {}	13	3	n	9
4	[]	50	4	yesno	3
5	for	10	5	error	3
6	if	3	6	y	5
7	>	1	7	coef	6
8	<	10	8	a	20
9	+	1	9	det	3
10	++	10	10	i	36
11	-	6	11	j	20
12	*	15	12	b	10
13	return	2	13	0	22
14	/	1	14	1	14
15	!	1	15	2	12
16	!=	2	16	true	1
17	while	1	17	false	1
18	==	1	18	3	2
19	get_data	1			
20	write_data	1			
21	solve	1			
22	setup	3			
23	deter	2			
Всего		199	Всего		176

Таблица 4 – Автоматический расчёт характеристик программы на С

№	Оператор	Количество	№	Операнд	Количество
1	!=	2	1	0	24
2	()	23	2	1	15
3	*	15	3	2	12
4	+	1	4	CMAX	4
5	++	10	5	RMAX	5
6	,	20	6	a	25
7	-	6	7	b	12
8	/	1	8	coef	8
9	;	94	9	det	4
10	<	10	10	i	40
11	=	24	11	j	22
12	==	2	12	n	10
13	>	1	13	error	4
14	sizeof	6	14	y	7
15	[]	51	15	yesno	4
16	_&	3			
17	__*	24			
18	deter	3			
19	do while	1			
20	for	10			
21	get_data	2			
22	if	3			
23	main	1			
24	malloc	6			
25	return	2			
26	setup	4			
27	solve	2			
28	write_data	2			
29		1			
Всего		330	Всего		196

Результаты расчета метрик также представлены в табл. 6.

Далее проведем ручной подсчет для программы на Ассемблере. Результат подсчета представлен в табл. 5, а результаты расчета метрик – в табл. 6.

Таблица 5 – Ручной расчёт характеристик программы на Ассемблере

№	Оператор	Количество	№	Операнд	Количество
1	pushq	7	1	%rbp	12
2	movq	88	2	%rsp	13
3	subq	7	3	\$48	1
4	movl	73	4	%rdi	21
5	jmp .L2	1	5	-24(%rbp)	16
6	leaq	24	6	%rsi	9
7	jmp .L3	1	7	\$10	6
8	cltq	9	8	\$3	13
9	addq	44	9	n(%rip)	9
10	movslq	12	10	\$0	24
11	salq	12	11	-8(%rbp)	27
12	addl	10	12	%eax	47
13	cmpl	11	13	%esi	2
14	jl .L4	1	14	.LC0(%rip)	1
15	jl .L5	1	15	-4(%rbp)	20
16	jmp .L6	1	16	.LC1(%rip)	1
17	jmp .L7	1	17	0(,%rax,8)	9
18	movsd	27	18	%rax	155
19	jl .L8	1	19	\$3	14
20	jl .L9	1	20	.LC2(%rip)	1
21	nop	4	21	\$1	17
22	leave	3	22	.LC3(%rip)	1
23	ret	6	23	-32(%rbp)	14
24	jmp .L11	1	24	.LC4(%rip)	1
25	jl .L12	1	25	%edi	12
26	mulsd	9	26	-40(%rbp)	4
27	subsd	4	27	%xmm0	29
28	movapd	3	28	.LC5(%rip)	2
29	popq	5	29	.LC6(%rip)	1

30	jmp .L16	1	30	\$32	1
31	divsd	1	31	coef(%rip)	5
32	call deter	2	32	%xmm2	12
33	call malloc@PLT	6	33	\$8	14
34	jle .L21	1	34	%xmm1	21
35	jmp .L22	1	35	\$16	11
36	movb	2	36	%xmm3	6
37	jmp .L23	1	37	%xmm4	4
38	jl .L24	1	38	-36(%rbp)	6
39	jl .L25	1	39	\$40	1
40	pxor	2	40	y(%rip)	3
41	ucomisd	2	41	%ecx	6
42	jp .L26	1	42	%rcx	22
43	jne .L26	1	43	a(%rip)	5
44	jmp .L30	1	44	-28(%rbp)	5
45	call setup	3	45	\$2	3
46	jmp .L32	1	46	\$24	10
47	jle .L33	1	47	error(%rip)	3
48	call get_data	1	48	.LC8(%rip)	1
49	call solve	1	49	%rbx	8
50	call write	1	50	.LC9(%rip)	1
51	movzbl	3	51	%al	4
52	xorl	1	52	.LC10(%rip)	1
53	testb	1	53	yesno(%rip)	3
54	je .L34	34	54	.LC11(rip)	1
55	cmpb	2	55	\$89	1
			56	\$121	1
Всего		403	Всего		641

Далее произведем расчет метрик Холстеда по имеющимся данным. Результаты для всех языков и вариантов измерения представлены в табл. 6.

Таблица 6 – Результаты расчета метрик

Характеристика	Ручной расчёт Pascal	Программный расчёт Pascal	Ручной расчёт Си	Программный расчёт Си	Ручной расчёт Ассемблер
Число простых операторов n_1	20	25	23	29	53
Число простых операндов n_2	19	27	18	15	56
Общее число всех операторов N_1	131	233	199	330	403
Общее число всех операндов N_2	140	183	176	196	641
Словарь n	39	52	41	44	109
Длина $N_{\text{опыт}}$	271	416	375	526	1044
Теоретическая длина $N_{\text{теор}}$	167,15	244,48	179,10	199,48	628,79
Объём V	1432,34	2371,38	2009,08	2871,66	7065,98
Потенциальный объём V^*	8	8	8	8	8
Уровень программы	0,005585	0,003374	0,003982	0,002786	0,001132
Оценка уровня программы L^{\sim}	0,013571	0,011803	0,008893	0,005278	0,003297
Интеллектуальное содержание I	19,43895	27,99009	17,86733	15,15655	23,29468
Работа программирования E	256451,2	702932,1	504551,3	1030805	6241017
Оценка времени программирования T^{\wedge}	25645,12	70293,21	50455,13	103080,5	624101,7
Время программирования T	10554,11	20090,88	22591,01	54408,4	214332,8
Уровень языка λ	0,044682	0,026988	0,031855	0,022287	0,009057
Ожидаемое число ошибок в программе B	2	3	3	3	8

Проведем сравнение полученных результатов. Как видно из таблицы, самый низкий уровень у программы на Ассемблере с учетом оценки. Интеллектуальное содержание выше у тех программ, которые были написаны более опытными программистами (Pascal) или сгенерированы автоматически (Ассемблер).

Как и ожидалось, примерное число ошибок растет с увеличением трудозатрат и времени на программу.

Код программ на Pascal, C и Ассемблере представлен в приложениях А, Б и В соответственно.

Выводы.

В ходе выполнения данной работы были на практике изучены методы расчета метрических характеристик качества разработки программ на основе метрик Холстеда. Также был произведен сравнительный анализ результатов для языков Pascal, C и Ассемблер.

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;
    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];
        end;
        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;
        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 ПРОГРММЫ НА С

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

#define RMAX 3
#define CMAX 3

int n;
char yesno;
int oshibka;

double* y;
double* coef;
double** a;

double det;

void get_data(double** a, double* y){
    int i,j;
    printf("\n");
    n = RMAX;

    for(i=0; i<n; i++){
        printf(" Equation %d", i);
        for(j=0; j<n; j++){
            printf("%d:", j);
            scanf("%lf", &a[i][j]);
        }
        printf(", C:");
        scanf("%lf\n", &y[i]);
    }
    printf("\n");
    for(i=0; i<n; i++){
        for(j=0; j<n; j++)
            printf("%lf ", a[i][j]);
        printf(":%lf\n", y[i]);
    }
    printf("\n");
}

void write_data(){
    int i;
    for (i=0; i<n; i++)
        printf("%lf ", coef[i]);
    printf("\n");
}

double deter(double** 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(double** b, double* coef, int j){
    int i;
```

```

        for(i=0; i<n; i++){
            b[i][j] = y[i];
            if (j>0)
                b[i][j-1] = a[i][j-1];
        }
        coef[j] = deter(b) / det;
    }

void solve(){
    double** b;
    b = (double**)malloc(RMAX*sizeof(double*));
    int i,j;
    for (i=0; i<RMAX; i++)
        b[i] = (double*)malloc(CMAX*sizeof(double));

    oshibka = 1;
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            b[i][j] = a[i][j];
    det = deter(b);
    if (det==0){
        oshibka = 0;
        printf("ERROR: matrix is singular.");
    } else {
        setup(b, coef, 0);
        setup(b, coef, 1);
        setup(b, coef, 2);
    }
}

int main()
{
    y = (double*)malloc(CMAX*sizeof(double));
    coef = (double*)malloc(CMAX*sizeof(double));

    a = (double**)malloc(RMAX*sizeof(double*));
    int i;
    for (i=0; i<RMAX; i++)
        a[i] = (double*)malloc(CMAX*sizeof(double));

    printf("\n");
    printf("Simultaneous soulution by Cramers rule");
    do{
        get_data(a, y);
        solve();
        if (oshibka == 0)
            write_data();
        printf("\n");
        printf("More?\n");
        scanf("%c", &yesno);
    } while (yesno != 'Y' || yesno != 'y');

    return 0;
}

```

ПРИЛОЖЕНИЕ В

КОД ПРОГРАММЫ НА АССЕМБЛЕРЕ

```

.file "prog.c"
.text
.comm n,4,4
.comm yesno,1,1
.comm error,1,1
.comm y,8,8
.comm coef,8,8
.comm a,8,8
.comm det,8,8
.section .rodata
.LC0:
.string " Equation %d"
.LC1:
.string "%d:"
.LC2:
.string "%lf"
.LC3:
.string ", C:"
.LC4:
.string "%lf\n"
.LC5:
.string "%lf "
.LC6:
.string ":%lf\n"
.text
.globl get_data
.type get_data, @function
get_data:
.LFB5:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $48, %rsp
movq %rdi, -24(%rbp)
movq %rsi, -32(%rbp)
movl $10, %edi
call putchar@PLT
movl $3, n(%rip)
movl $0, -8(%rbp)
jmp .L2
.L5:
movl -8(%rbp), %eax
movl %eax, %esi
leaq .LC0(%rip), %rdi
movl $0, %eax
call printf@PLT
movl $0, -4(%rbp)
jmp .L3
.L4:
movl -4(%rbp), %eax
movl %eax, %esi
leaq .LC1(%rip), %rdi
movl $0, %eax
call printf@PLT
movl -8(%rbp), %eax

```



```

        cltq
        leaq    0(,%rax,8), %rdx
        movq    -24(%rbp), %rax
        addq    %rdx, %rax
        movq    (%rax), %rax
        movl    -4(%rbp), %edx
        movslq  %edx, %rdx
        salq    $3, %rdx
        addq    %rdx, %rax
        movq    %rax, %rsi
        leaq    .LC2(%rip), %rdi
        movl    $0, %eax
        call    __isoc99_scanf@PLT
        addl    $1, -4(%rbp)
.L3:
        movl    n(%rip), %eax
        cmpl    %eax, -4(%rbp)
        jl      .L4
        leaq    .LC3(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    -8(%rbp), %eax
        cltq
        leaq    0(,%rax,8), %rdx
        movq    -32(%rbp), %rax
        addq    %rdx, %rax
        movq    %rax, %rsi
        leaq    .LC4(%rip), %rdi
        movl    $0, %eax
        call    __isoc99_scanf@PLT
        addl    $1, -8(%rbp)
.L2:
        movl    n(%rip), %eax
        cmpl    %eax, -8(%rbp)
        jl      .L5
        movl    $10, %edi
        call    putchar@PLT
        movl    $0, -8(%rbp)
        jmp     .L6
.L9:
        movl    $0, -4(%rbp)
        jmp     .L7
.L8:
        movl    -8(%rbp), %eax
        cltq
        leaq    0(,%rax,8), %rdx
        movq    -24(%rbp), %rax
        addq    %rdx, %rax
        movq    (%rax), %rax
        movl    -4(%rbp), %edx
        movslq  %edx, %rdx
        salq    $3, %rdx
        addq    %rdx, %rax
        movq    (%rax), %rax
        movq    %rax, -40(%rbp)
        movsd   -40(%rbp), %xmm0
        leaq    .LC5(%rip), %rdi
        movl    $1, %eax
        call    printf@PLT
        addl    $1, -4(%rbp)
.L7:

```

```

        movl    n(%rip), %eax
        cmpl    %eax, -4(%rbp)
        jl      .L8
        movl    -8(%rbp), %eax
        cltq
        leaq    0(,%rax,8), %rdx
        movq    -32(%rbp), %rax
        addq    %rdx, %rax
        movq    (%rax), %rax
        movq    %rax, -40(%rbp)
        movsd   -40(%rbp), %xmm0
        leaq    .LC6(%rip), %rdi
        movl    $1, %eax
        call    printf@PLT
        addl    $1, -8(%rbp)
.L6:
        movl    n(%rip), %eax
        cmpl    %eax, -8(%rbp)
        jl      .L9
        movl    $10, %edi
        call    putchar@PLT
        nop
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE5:
        .size   get_data, .-get_data
        .globl  write_data
        .type   write_data, @function
write_data:
.LFB6:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $32, %rsp
        movl    $0, -4(%rbp)
        jmp     .L11
.L12:
        movq    coef(%rip), %rax
        movl    -4(%rbp), %edx
        movslq   %edx, %rdx
        salq    $3, %rdx
        addq    %rdx, %rax
        movq    (%rax), %rax
        movq    %rax, -24(%rbp)
        movsd   -24(%rbp), %xmm0
        leaq    .LC5(%rip), %rdi
        movl    $1, %eax
        call    printf@PLT
        addl    $1, -4(%rbp)
.L11:
        movl    n(%rip), %eax
        cmpl    %eax, -4(%rbp)
        jl      .L12
        movl    $10, %edi
        call    putchar@PLT
        nop

```

```

        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE6:
        .size write_data, .-write_data
        .globl deter
        .type deter, @function
deter:
.LFB7:
        .cfi_startproc
        pushq %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq %rsp, %rbp
        .cfi_def_cfa_register 6
        movq %rdi, -8(%rbp)
        movq -8(%rbp), %rax
        movq (%rax), %rax
        movsd (%rax), %xmm2
        movq -8(%rbp), %rax
        addq $8, %rax
        movq (%rax), %rax
        addq $8, %rax
        movsd (%rax), %xmm1
        movq -8(%rbp), %rax
        addq $16, %rax
        movq (%rax), %rax
        addq $16, %rax
        movsd (%rax), %xmm0
        mulsd %xmm1, %xmm0
        movq -8(%rbp), %rax
        addq $16, %rax
        movq (%rax), %rax
        addq $8, %rax
        movsd (%rax), %xmm3
        movq -8(%rbp), %rax
        addq $8, %rax
        movq (%rax), %rax
        addq $16, %rax
        movsd (%rax), %xmm1
        mulsd %xmm3, %xmm1
        subsd %xmm1, %xmm0
        mulsd %xmm2, %xmm0
        movq -8(%rbp), %rax
        movq (%rax), %rax
        addq $8, %rax
        movsd (%rax), %xmm3
        movq -8(%rbp), %rax
        addq $8, %rax
        movq (%rax), %rax
        movsd (%rax), %xmm2
        movq -8(%rbp), %rax
        addq $16, %rax
        movq (%rax), %rax
        addq $16, %rax
        movsd (%rax), %xmm1
        mulsd %xmm2, %xmm1
        movq -8(%rbp), %rax
        addq $16, %rax
        movq (%rax), %rax

```

```

movsd (%rax), %xmm4
movq -8(%rbp), %rax
addq $8, %rax
movq (%rax), %rax
addq $16, %rax
movsd (%rax), %xmm2
mulsd %xmm4, %xmm2
subsd %xmm2, %xmm1
mulsd %xmm3, %xmm1
subsd %xmm1, %xmm0
movapd %xmm0, %xmm1
movq -8(%rbp), %rax
movq (%rax), %rax
addq $16, %rax
movsd (%rax), %xmm3
movq -8(%rbp), %rax
addq $8, %rax
movq (%rax), %rax
movsd (%rax), %xmm2
movq -8(%rbp), %rax
addq $16, %rax
movq (%rax), %rax
addq $8, %rax
movsd (%rax), %xmm0
mulsd %xmm2, %xmm0
movq -8(%rbp), %rax
addq $16, %rax
movq (%rax), %rax
movsd (%rax), %xmm4
movq -8(%rbp), %rax
addq $8, %rax
movq (%rax), %rax
addq $8, %rax
movsd (%rax), %xmm2
mulsd %xmm4, %xmm2
subsd %xmm2, %xmm0
mulsd %xmm3, %xmm0
addsd %xmm1, %xmm0
popq %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE7:
.size deter, .-deter
.globl setup
.type setup, @function
setup:
.LFB8:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $40, %rsp
movq %rdi, -24(%rbp)
movq %rsi, -32(%rbp)
movl %edx, -36(%rbp)
movl $0, -4(%rbp)
jmp .L16
.L18:

```

```

movq y(%rip), %rax
movl -4(%rbp), %edx
movslq %edx, %rdx
salq $3, %rdx
addq %rax, %rdx
movl -4(%rbp), %eax
cltq
leaq 0(,%rax,8), %rcx
movq -24(%rbp), %rax
addq %rcx, %rax
movq (%rax), %rax
movl -36(%rbp), %ecx
movslq %ecx, %rcx
salq $3, %rcx
addq %rcx, %rax
movsd (%rdx), %xmm0
movsd %xmm0, (%rax)
cmpl $0, -36(%rbp)
jle .L17
movq a(%rip), %rax
movl -4(%rbp), %edx
movslq %edx, %rdx
salq $3, %rdx
addq %rdx, %rax
movq (%rax), %rax
movl -36(%rbp), %edx
movslq %edx, %rdx
salq $3, %rdx
subq $8, %rdx
addq %rax, %rdx
movl -4(%rbp), %eax
cltq
leaq 0(,%rax,8), %rcx
movq -24(%rbp), %rax
addq %rcx, %rax
movq (%rax), %rax
movl -36(%rbp), %ecx
movslq %ecx, %rcx
salq $3, %rcx
subq $8, %rcx
addq %rcx, %rax
movsd (%rdx), %xmm0
movsd %xmm0, (%rax)
.L17:
addl $1, -4(%rbp)
.L16:
movl n(%rip), %eax
cmpl %eax, -4(%rbp)
jl .L18
movq -24(%rbp), %rax
movq %rax, %rdi
call deter
movapd %xmm0, %xmm1
movsd det(%rip), %xmm0
movl -36(%rbp), %eax
cltq
leaq 0(,%rax,8), %rdx
movq -32(%rbp), %rax
addq %rdx, %rax
divsd %xmm0, %xmm1
movapd %xmm1, %xmm0

```

```

        movsd %xmm0, (%rax)
        nop
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE8:
        .size setup, .-setup
        .section      .rodata
.LC8:
        .string      "ERROR: matrix is singular."
        .text
        .globl solve
        .type solve, @function
solve:
.LFB9:
        .cfi_startproc
        pushq %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq %rsp, %rbp
        .cfi_def_cfa_register 6
        pushq %rbx
        subq $24, %rsp
        .cfi_offset 3, -24
        movl $24, %edi
        call malloc@PLT
        movq %rax, -24(%rbp)
        movl $0, -32(%rbp)
        jmp .L20
.L21:
        movl -32(%rbp), %eax
        cltq
        leaq 0(,%rax,8), %rdx
        movq -24(%rbp), %rax
        leaq (%rdx,%rax), %rbx
        movl $24, %edi
        call malloc@PLT
        movq %rax, (%rbx)
        addl $1, -32(%rbp)
.L20:
        cmpl $2, -32(%rbp)
        jle .L21
        movb $0, error(%rip)
        movl $0, -32(%rbp)
        jmp .L22
.L25:
        movl $0, -28(%rbp)
        jmp .L23
.L24:
        movq a(%rip), %rax
        movl -32(%rbp), %edx
        movslq %edx, %rdx
        salq $3, %rdx
        addq %rdx, %rax
        movq (%rax), %rax
        movl -28(%rbp), %edx
        movslq %edx, %rdx
        salq $3, %rdx
        addq %rax, %rdx
        movl -32(%rbp), %eax

```

```

        cltq
        leaq    0(,%rax,8), %rcx
        movq    -24(%rbp), %rax
        addq    %rcx, %rax
        movq    (%rax), %rax
        movl    -28(%rbp), %ecx
        movslq  %ecx, %rcx
        salq    $3, %rcx
        addq    %rcx, %rax
        movsd   (%rdx), %xmm0
        movsd   %xmm0, (%rax)
        addl    $1, -28(%rbp)
.L23:
        movl    n(%rip), %eax
        cmpl    %eax, -28(%rbp)
        jl      .L24
        addl    $1, -32(%rbp)
.L22:
        movl    n(%rip), %eax
        cmpl    %eax, -32(%rbp)
        jl      .L25
        movq    -24(%rbp), %rax
        movq    %rax, %rdi
        call    deter
        movq    %xmm0, %rax
        movq    %rax, det(%rip)
        movsd   det(%rip), %xmm0
        pxor    %xmm1, %xmm1
        ucomisd  %xmm1, %xmm0
        jp      .L26
        pxor    %xmm1, %xmm1
        ucomisd  %xmm1, %xmm0
        jne     .L26
        movb    $1, error(%rip)
        leaq    .LC8(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        jmp     .L30
.L26:
        movq    coef(%rip), %rcx
        movq    -24(%rbp), %rax
        movl    $0, %edx
        movq    %rcx, %rsi
        movq    %rax, %rdi
        call    setup
        movq    coef(%rip), %rcx
        movq    -24(%rbp), %rax
        movl    $1, %edx
        movq    %rcx, %rsi
        movq    %rax, %rdi
        call    setup
        movq    coef(%rip), %rcx
        movq    -24(%rbp), %rax
        movl    $2, %edx
        movq    %rcx, %rsi
        movq    %rax, %rdi
        call    setup
.L30:
        nop
        addq    $24, %rsp
        popq    %rbx

```

```

        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE9:
        .size   solve, .-solve
        .section .rodata
        .align 8
.LC9:
        .string  "Simultaneous soulution by Cramers rule"
.LC10:
        .string  "More?"
.LC11:
        .string  "%c"
        .text
        .globl main
        .type    main, @function
main:
.LFB10:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        pushq   %rbx
        subq    $24, %rsp
        .cfi_offset 3, -24
        movl    $24, %edi
        call    malloc@PLT
        movq    %rax, y(%rip)
        movl    $24, %edi
        call    malloc@PLT
        movq    %rax, coef(%rip)
        movl    $24, %edi
        call    malloc@PLT
        movq    %rax, a(%rip)
        movl    $0, -20(%rbp)
        jmp     .L32
.L33:
        movq    a(%rip), %rax
        movl    -20(%rbp), %edx
        movslq  %edx, %rdx
        salq    $3, %rdx
        leaq    (%rax,%rdx), %rbx
        movl    $24, %edi
        call    malloc@PLT
        movq    %rax, (%rbx)
        addl    $1, -20(%rbp)
.L32:
        cmpl    $2, -20(%rbp)
        jle     .L33
        movl    $10, %edi
        call    putchar@PLT
        leaq    .LC9(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
.L35:
        movq    y(%rip), %rdx
        movq    a(%rip), %rax
        movq    %rdx, %rsi

```



```

        movq    %rax, %rdi
        call    get_data
        movl    $0, %eax
        call    solve
        movzbl error(%rip), %eax
        xorl    $1, %eax
        testb   %al, %al
        je      .L34
        movl    $0, %eax
        call    write_data
.L34:
        movl    $10, %edi
        call    putchar@PLT
        leaq    .LC10(%rip), %rdi
        call    puts@PLT
        leaq    yesno(%rip), %rsi
        leaq    .LC11(%rip), %rdi
        movl    $0, %eax
        call    __isoc99_scanf@PLT
        movzbl yesno(%rip), %eax
        cmpb    $89, %al
        jne     .L35
        movzbl yesno(%rip), %eax
        cmpb    $121, %al
        jne     .L35
        movl    $0, %eax
        addq    $24, %rsp
        popq    %rbx
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE10:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
        .section .note.GNU-stack,"",@progbits

```