

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

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

Студент гр. 7304

Кошманов Н.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2021

## **Цель работы**

Изучение и сравнение метрик Холстеда для программ на C, Pascal и ассемблере.

## **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.

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  $\square 2^*$  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:
  - а. Измеримые характеристики,
  - б. Расчетные характеристики
2. Паскаль. Программный расчет:
  - а) Измеримые характеристики,
  - б) Расчетные характеристики
3. Си. Ручной расчет:
  - а. Измеримые характеристики,
  - б. Расчетные характеристики
4. Си. Программный расчет:

- а. Измеримые характеристики,
  - б. Расчетные характеристики
- 5. Ассемблер. Ручной расчет:**
- а. Измеримые характеристики,
  - б. Расчетные характеристики
- 6. Сводная таблица расчетов для трех языков.**

## Ход работы

### 1. Вариант – 8, Умножение матриц.

Исходный код программы представлен в приложении А. В данной реализации исключены операторы, обеспечивающие интерфейс с пользователем и выдачу текстовых сообщений. Исправленная версия программы представлена в приложении Б.

### 2. Ручной расчет метрик программы на Pascal

Для программы из Приложения Б был произведен подсчет операторов и операндов в программе, а также расчет измеримых и расчетных характеристик программы. Результаты представлены в табл. 1, 2.

Таблица 1 — Операторы и операнды программы на Pascal

Оператор	Количество	Операнд	Количество
<b>square</b>	2	<b>i</b>	12
<b>get_data</b>	2	<b>l</b>	8
<b>if...then</b>	1	<b>k</b>	13
<b>&lt;&gt;</b>	1	<b>x</b>	10
<b>begin...end</b>	7	<b>a</b>	8
<b>for to do</b>	6	<b>g</b>	6
<b>+</b>	2	<b>j</b>	7
<b>*</b>	4	<b>ncol</b>	8
<b>;</b>	20	<b>nrow</b>	7

-	1	y	6
0	2	1	8
:=	16	2	2
П	16	0	2

Таблица 2 — Измеримые и расчетные характеристики для Pascal (ручной подсчет)

Метрика	Описание	Значение
$\eta_1$	Число простых (уникальных) операторов	13
$\eta_2$	Число простых (уникальных) операндов	13
$N_1$	Общее число всех операторов	80
$N_2$	Общее число всех операндов	97
$\eta$	Словарь, $\eta_1 + \eta_2$	26
$N$	Опытная длина, $N_1 + N_2$	177
$N_{\text{теор}}$	Теоретическая длина, $\eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$	96,211
$V$	Объем, $N \log_2 \eta$	831,978
$V^*$	Потенциальный объем, $(\eta_2^* + 2) \log_2 (\eta_2^* + 2)$ , $\eta_2^* = 6$	24
$L$	Уровень, $V^* / V$	0,0288
$I$	Интеллектуальное содержание, $2 / \eta_1 * \eta_2 / N_2 * N * \log_2 (\eta)$	17,15
$E$	Работа по программированию, $V / L$	28841,12
$\check{T}$	Время программирования, $E / S$ , $S = 10$	2884,112
$\lambda$	Уровень языка, $L * V^*$	0,692326
$B$	Количество ошибок, $V^* / L / 1000$	1

### 3. Программный расчет метрик программы на Pascal

Результаты представлены в табл. 3.

Таблица 3 — Измеримые и расчетные характеристики для Pascal (программный подсчет)

Метрика	Описание	Значение
$\eta_1$	Число простых (уникальных) операторов	14
$\eta_2$	Число простых (уникальных) операндов	23
$N_1$	Общее число всех операторов	61
$N_2$	Общее число всех операндов	120
$\eta$	Словарь, $\eta_1 + \eta_2$	37
$N$	Опытная длина, $N_1 + N_2$	181
$N_{\text{теор}}$	Теоретическая длина, $\eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$	157,345
$V$	Объем, $N \log_2 \eta$	942,911
$V^*$	Потенциальный объем, $(\eta_2^* + 2) \log_2 (\eta_2^* + 2)$ , $\eta_2^* = 6$	24
$L$	Уровень, $V^* / V$	0,02545
$I$	Интеллектуальное содержание, $2 / \eta_1^* \eta_2 / N_2^* N^* \log_2 (\eta)$	25,8178
$E$	Работа по программированию, $V / L$	37045,1
$\dot{T}$	Время программирования, $E / S$ , $S = 10$	3704,51
$\lambda$	Уровень языка, $L^* V^*$	0.610874
$B$	Количество ошибок, $V^* / L / 1000$	1

#### 4. Ручной расчет метрик программы на С

Для программы из Приложения Б была написана программа на языке С, был произведен подсчет операторов и операндов в программе на С, а также расчет измеримых и расчетных характеристик программы. Результаты представлены в табл. 4, 5

Таблица 4 — Операторы и операнды программы на С

Оператор	Количество	Операнд	Количество
;	23	i	20

=	18	l	10
[] or [][]	16	k	15
()	11	x	11
{}	9	a	7
++	6	g	6
for	6	j	5
<	6	ncol	7
*	4	nrow	8
pointer*	2	y	7
+	4	1	5
square	2	2	1
get_data	2	0	9
!=	1		
-	1		
<=	1		
if	1		

Таблица 5 — Измеримые и расчетные характеристики для С (ручной подсчет)

Метрика	Описание	Значение
$\eta_1$	Число простых (уникальных) операторов	17
$\eta_2$	Число простых (уникальных) операндов	13
$N_1$	Общее число всех операторов	113
$N_2$	Общее число всех операндов	111
$\eta$	Словарь, $\eta_1 + \eta_2$	30
$N$	Опытная длина, $N_1 + N_2$	224
$N_{\text{теор}}$	Теоретическая длина, $\eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$	117,5926
$V$	Объем, $N \log_2 \eta$	1099,143
$V^*$	Потенциальный объем, $(\eta_2^* + 2) \log_2 (\eta_2^* + 2)$ , $\eta_2^*=6$	24



<b>L</b>	Уровень, $V^*/V$	0,0218
<b>I</b>	Интеллектуальное содержание, $2/\eta_1 * \eta_2 / N_2 * N * \log_2(\eta)$	15,144
<b>E</b>	Работа по программированию, $V/L$	50338,18
<b>T</b>	Время программирования, $E/S, S=10$	5033,818
$\lambda$	Уровень языка, $L * V^*$	0,52404
<b>B</b>	Количество ошибок, $V^*/L/1000$	2

## 5. Программный расчет метрик программы на С

Результаты представлены в табл. 6.

Таблица 6 — Измеримые и расчетные характеристики для С (программный подсчет)

Метрика	Описание	Значение
$\eta_1$	Число простых (уникальных) операторов	20
$\eta_2$	Число простых (уникальных) операндов	17
$N_1$	Общее число всех операторов	122
$N_2$	Общее число всех операндов	119
$\eta$	Словарь, $\eta_1 + \eta_2$	37
$N$	Опытная длина, $N_1 + N_2$	241
$N_{\text{теор}}$	Теоретическая длина, $\eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$	155,925
$V$	Объем, $N \log_2 \eta$	1255,48
$V^*$	Потенциальный объем, $(\eta_2^* + 2) \log_2(\eta_2^* + 2), \eta_2^*=6$	24
<b>L</b>	Уровень, $V^*/V$	0,01911
<b>I</b>	Интеллектуальное содержание, $2/\eta_1 * \eta_2 / N_2 * N * \log_2(\eta)$	17,934
<b>E</b>	Работа по программированию, $V/L$	65676,1
<b>T</b>	Время программирования, $E/S, S=10$	6567,61
$\lambda$	Уровень языка, $L * V^*$	0.458789
<b>B</b>	Количество ошибок, $V^*/L/1000$	2

## 6. Ручной расчет метрик программы на Assembly

Для программы из Приложения В был создан код на языке Assembly, используя gcc -S program.c. Был произведен подсчет операторов и операндов в программе на Assembly, а также расчет измеримых и расчетных характеристик программы. Результаты представлены в табл. 7, 7

Таблица 7 — Операторы и операнды программы на Assembly

Оператор	Количество	Операнд	Количество
<b>addl</b>	8	<b>\$0</b>	6
<b>addq</b>	37	<b>\$1</b>	10
<b>addsd</b>	2	<b>\$3</b>	12
<b>call</b>	3	<b>\$5</b>	1
<b>cltq</b>	15	<b>\$432</b>	1
<b>cmpl</b>	7	<b>%eax</b>	60
<b>cvtsi2sd</b>	2	<b>%ecx</b>	4
<b>get_data</b>	1	<b>%edi</b>	2
<b>get_data:</b>	1	<b>%edx</b>	2
<b>je</b>	2	<b>%r8d</b>	4
<b>jl</b>	5	<b>%r9d</b>	2
<b>jle</b>	1	<b>%rax</b>	103
<b>jmp</b>	6	<b>%rbp</b>	8
<b>leal</b>	1	<b>%rcx</b>	3
<b>leaq</b>	12	<b>%rdi</b>	6
<b>leave</b>	1	<b>%rdx</b>	66
<b>main:</b>	1	<b>%rsi</b>	4
<b>movl</b>	54	<b>%rsp</b>	4
<b>movq</b>	52	<b>%xmm0</b>	23
<b>movsd</b>	17	<b>%xmm1</b>	6

<b>movslq</b>	11	<b>%xmm2</b>	4
<b>mulsd</b>	3	<b>(%rax%rax)</b>	1
<b>nop</b>	2	<b>(%rax)</b>	6
<b>popq</b>	2	<b>(%rcx%rax8)</b>	1
<b>pushq</b>	3	<b>(%rdx%rax)</b>	1
<b>pxor</b>	2	<b>(%rdx%rax8)</b>	9
<b>ret</b>	3	<b>-12(%rbp)</b>	10
<b>salq</b>	11	<b>-16(%rbp)</b>	15
<b>square</b>	1	<b>-224(%rbp)</b>	2
<b>square:</b>	1	<b>-24(%rbp)</b>	8
<b>subl</b>	1	<b>-304(%rbp)</b>	1
<b>subq</b>	1	<b>-32(%rbp)</b>	4
<b>xorl</b>	1	<b>-36(%rbp)</b>	2
<b>xorq</b>	1	<b>-384(%rbp)</b>	2
		<b>-4(%rbp)</b>	10
		<b>-40(%rbp)</b>	8
		<b>-416(%rbp)</b>	1
		<b>-420(%rbp)</b>	3
		<b>-424(%rbp)</b>	3
		<b>-48(%rbp)</b>	4
		<b>-52(%rbp)</b>	3
		<b>-56(%rbp)</b>	2
		<b>-8(%rbp)</b>	16
		<b>.LC0(%rip)</b>	1
		<b>0(%rax8)</b>	1

Таблица 8 — Измеримые и расчетные характеристики для Assembly (ручной подсчет)

Метрика	Описание	Значение
$\eta_1$	Число простых (уникальных) операторов	34
$\eta_2$	Число простых (уникальных) операндов	45
$N_1$	Общее число всех операторов	271
$N_2$	Общее число всех операндов	445
$\eta$	Словарь, $\eta_1 + \eta_2$	79
$N$	Опытная длина, $N_1 + N_2$	716
$N_{\text{теор}}$	Теоретическая длина, $\eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$	420,1071
$V$	Объем, $N \log_2 \eta$	4513,507
$V^*$	Потенциальный объем, $(\eta_2^* + 2) \log_2 (\eta_2^* + 2)$ , $\eta_2^* = 6$	24
$L$	Уровень, $V^* / V$	0,005317
$I$	Интеллектуальное содержание, $2 / \eta_1^* \eta_2 / N_2^* N^* \log_2 (\eta)$	26,84836
$E$	Работа по программированию, $V / L$	848822,7
$T$	Время программирования, $E / S$ , $S = 10$	84882,27
$\lambda$	Уровень языка, $L^* V^*$	0,1276
$B$	Количество ошибок, $V^* / L / 1000$	5

## 7. Сводная таблица результатов

Для языков Pascal, C, Assembly составлена сводная таблица 9 по измеримым и расчетным характеристикам.

Таблица 9 — Сводная таблица расчетов

	Pascal		C		Assembly
	Ручной	Программный	Ручной	Программный	Ручной
$\eta_1$	13	14	17	20	34
$\eta_2$	13	23	13	17	45
$N_1$	80	61	113	122	271

<b>N<sub>2</sub></b>	97	120	111	119	445
<b>η</b>	26	37	30	37	79
<b>N</b>	177	181	224	241	716
<b>N<sub>теор</sub></b>	96,211	157,345	117,5926	155,925	420,1071
<b>V</b>	831,978	942,911	1099,143	1255,48	4513,507
<b>V*</b>	24	24	24	24	24
<b>L</b>	0,0288	0,02545	0,0218	0,01911	0,005317
<b>I</b>	17,15	25,8178	15,144	17,934	26,84836
<b>E</b>	28841,12	37045,1	50338,18	65676,1	848822,7
<b>T</b>	2884,112	3704,51	5033,818	6567,61	84882,27
<b>λ</b>	0,692326	0.610874	0,52404	0.458789	0,1276
<b>B</b>	1	1	2	2	5

## Вывод

В результате выполнения данной лабораторной работы была изучена система метрик Холстеда. Было проведено сравнение программ на языках Pascal, Си и Ассемблер

## ПРИЛОЖЕНИЕ А

### Исходный код на Pascal

```
program matr1;
{ pascal program to perform matrix multiplication }

const  rmax  = 9;
       cmax  = 3;

type   ary   = array[1..rmax] of real;
       arys  = array[1..cmax] of real;
       ary2  = array[1..rmax,1..cmax] of real;
       ary2s = array[1..cmax,1..cmax] of real;

var    y      : ary;
       g      : arys;
       x      : ary2;
       a      : ary2s;
       nrow,ncol : integer;

procedure get_data(var x: ary2;
                  var y: ary;
                  var nrow,ncol: integer);

{ get the values for nrow, ncol, and arrays x,y }

var    i,j    : integer;

begin
  nrow:=5; {эти значения не следует считать }
  ncol:=3; { фиксированными - они могут изменяться}
  for i:=1 to nrow do
    begin
      x[i,1]:=1;
      for j:=2 to ncol do
        x[i,j]:=i*x[i,j-1];
      y[i]:=2*i
    end
  end;
  { procedure get_data }
  procedure write_data;

  { print out the answeres }

  var
    i,j : integer;

  begin
    ClrScr;
    writeln;
    writeln('          X          Y');
    for i:=1 to nrow do
      begin
        for j:=1 to ncol do
          write(x[i,j]:7:1,' ');
        writeln(':',y[i]:7:1)
      end;
    writeln('          A          G');
    for i:=1 to ncol do
      begin
        for j:=1 to ncol do
```

```

        write(a[i,j]:7:1,' ');
        writeln(':',g[i]:7:1)
    end
end;      { write_data }

procedure square(x: ary2;
                y: ary;
                var a: ary2s;
                var g: arys;
                nrow,ncol: integer);

{ matrix multiplication routine }
{ a= transpose x times x }
{ g= y times x }

var
    i,k,l: integer;

begin      { square }
    for k:=1 to ncol do
        begin
            for l:=1 to k do
                begin
                    a[k,l]:=0;
                    for i:=1 to nrow do
                        begin
                            a[k,l]:=a[k,l]+x[i,l]*x[i,k];
                            if k>l then a[l,k]:=a[k,l]
                        end
                    end;      { l-loop }
                    g[k]:=0;
                    for i:=1 to nrow do
                        g[k]:=g[k]+y[i]*x[i,k]
                    end { k-loop }
                end; { square }
            end;
        end;

begin { MAIN program }
    get_data(x,y,nrow,ncol);
    square(x,y,a,g,nrow,ncol);
    write_data
end.

```

## ПРИЛОЖЕНИЕ Б

### Измененный код на Pascal

```
program matr1;

const
  rmax = 9;
  cmax = 3;

type
  ary = array[1..rmax] of real;
  arys = array[1..cmax] of real;
  ary2 = array[1..rmax, 1..cmax] of real;
  ary2s = array[1..cmax, 1..cmax] of real;

var
  y: ary;
  g: arys;
  x: ary2;
  a: ary2s;
  nrow, ncol: integer;

procedure get_data(var x: ary2; var y: ary; var nrow, ncol: integer);
var
  i, j: integer;

begin
  nrow := 5;
  ncol := 3;
  for i := 1 to nrow do
    begin
      x[i, 1] := 1;
      for j := 2 to ncol do
        x[i, j] := i * x[i, j - 1];
      y[i] := 2 * i;
    end;
  end;

procedure square(x: ary2; y: ary; var a: ary2s; var g: arys; nrow, ncol:
integer);
var
  i, k, l: integer;

begin
  for k := 1 to ncol do
    begin
      for l := 1 to k do
        begin
          a[k, l] := 0;
          for i := 1 to nrow do
            begin
              a[k, l] := a[k, l] + x[i, l] * x[i, k];
              if k <> l then
                a[l, k] := a[k, l];
            end;
          end;
          g[k] := 0;
          for i := 1 to nrow do
            g[k] := g[k] + y[i] * x[i, k];
          end;
        end;
      end;

begin
```



```
    get_data(x, y, nrow, ncol);  
    square(x, y, a, g, nrow, ncol);  
end.
```

## ПРИЛОЖЕНИЕ В

### Исходный код на С

```
#include <stdio.h>

#define rmax 9
#define cmax 3

typedef double ary[rmax];
typedef double arys[cmax];
typedef double ary2[rmax][cmax];
typedef double ary2s[cmax][cmax];
typedef double* pointer;
typedef double** dpointer;

void get_data(ary2 x, ary y, int nrow, int ncol) {
    for (int i = 0; i < nrow; i++) {
        x[i][0] = 1;
        for (int j = 1; j < ncol; j++) {
            x[i][j] = (i + 1) * x[i][j - 1];
        }
        y[i] = 2 * (i + 1);
    }
}

void square(ary2 x, double* y, ary2s a, double* g, int nrow, int ncol) {
    for (int k = 0; k < ncol; k++) {
        for (int l = 0; l <= k; l++) {
            a[k][l] = 0;

            for (int i = 0; i < nrow; i++) {
                a[k][l] = a[k][l] + x[i][l] * x[i][k];
                if (k != l)
                    a[l][k] = a[k][l];
            }

            g[k] = 0;
            for (int i = 0; i < nrow; i++) {
                g[k] = g[k] + y[i] * x[i][k];
            }
        }
    }
}

int main() {
    int nrow = 5;
    int ncol = 3;

    ary2 x;
    ary y;

    arys g;
    ary2s a;

    get_data(x, y, nrow, ncol);
    square(x, y, a, g, nrow, ncol);

    return 0;
}
```

## ПРИЛОЖЕНИЕ Г

### Исходный код на Assembly

```
get_data:
    pushq %rbp
    movq %rsp, %rbp
    movq %rdi, -24(%rbp)
    movq %rsi, -32(%rbp)
    movl %edx, -36(%rbp)
    movl %ecx, -40(%rbp)
    movl $0, -8(%rbp)
    jmp .L2
    movl -8(%rbp), %eax
    movslq %eax, %rdx
    movq %rdx, %rax
    addq %rax, %rax
    addq %rdx, %rax
    salq $3, %rax
    movq %rax, %rdx
    movq -24(%rbp), %rax
    addq %rdx, %rax
    movsd .LC0(%rip), %xmm0
    movsd %xmm0, (%rax)
    movl $1, -4(%rbp)
    jmp .L3
    movl -8(%rbp), %eax
    addl $1, %eax
    cvtsi2sd %eax, %xmm1
    movl -8(%rbp), %eax
    movslq %eax, %rdx
    movq %rdx, %rax
    addq %rax, %rax
    addq %rdx, %rax
    salq $3, %rax
    movq %rax, %rdx
    movq -24(%rbp), %rax
    addq %rax, %rdx
    movl -4(%rbp), %eax
    subl $1, %eax
    cltq
    movsd (%rdx,%rax,8), %xmm0
    movl -8(%rbp), %eax
    movslq %eax, %rdx
    movq %rdx, %rax
    addq %rax, %rax
    addq %rdx, %rax
    salq $3, %rax
    movq %rax, %rdx
    movq -24(%rbp), %rax
    addq %rax, %rdx
    mulsd %xmm1, %xmm0
    movl -4(%rbp), %eax
    cltq
    movsd %xmm0, (%rdx,%rax,8)
    addl $1, -4(%rbp)
    movl -4(%rbp), %eax
    cmpl -40(%rbp), %eax
    jl .L4
    movl -8(%rbp), %eax
    addl $1, %eax
    leal (%rax,%rax), %ecx
```

```

movl -8(%rbp), %eax
cltq
leaq 0(,%rax,8), %rdx
movq -32(%rbp), %rax
addq %rdx, %rax
cvttsi2sd %ecx, %xmm0
movsd %xmm0, (%rax)
addl $1, -8(%rbp)
movl -8(%rbp), %eax
cmpl -36(%rbp), %eax
jl .L5
nop
popq %rbp
ret

```

square:

```

pushq %rbp
movq %rsp, %rbp
movq %rdi, -24(%rbp)
movq %rsi, -32(%rbp)
movq %rdx, -40(%rbp)
movq %rcx, -48(%rbp)
movl %r8d, -52(%rbp)
movl %r9d, -56(%rbp)
movl $0, -16(%rbp)
jmp .L7
movl $0, -12(%rbp)
jmp .L8
movl -16(%rbp), %eax
movslq %eax, %rdx
movq %rdx, %rax
addq %rax, %rax
addq %rdx, %rax
salq $3, %rax
movq %rax, %rdx
movq -40(%rbp), %rax
addq %rax, %rdx
movl -12(%rbp), %eax
cltq
pxor %xmm0, %xmm0
movsd %xmm0, (%rdx,%rax,8)
movl $0, -8(%rbp)
jmp .L9
movl -16(%rbp), %eax
movslq %eax, %rdx
movq %rdx, %rax
addq %rax, %rax
addq %rdx, %rax
salq $3, %rax
movq %rax, %rdx
movq -40(%rbp), %rax
addq %rax, %rdx
movl -12(%rbp), %eax
cltq
movsd (%rdx,%rax,8), %xmm1
movl -8(%rbp), %eax
movslq %eax, %rdx
movq %rdx, %rax
addq %rax, %rax
addq %rdx, %rax
salq $3, %rax
movq %rax, %rdx

```

```

movq -24(%rbp), %rax
addq %rax, %rdx
movl -12(%rbp), %eax
cltq
movsd (%rdx,%rax,8), %xmm2
movl -8(%rbp), %eax
movslq %eax, %rdx
movq %rdx, %rax
addq %rax, %rax
addq %rdx, %rax
salq $3, %rax
movq %rax, %rdx
movq -24(%rbp), %rax
addq %rax, %rdx
movl -16(%rbp), %eax
cltq
movsd (%rdx,%rax,8), %xmm0
mulsd %xmm2, %xmm0
movl -16(%rbp), %eax
movslq %eax, %rdx
movq %rdx, %rax
addq %rax, %rax
addq %rdx, %rax
salq $3, %rax
movq %rax, %rdx
movq -40(%rbp), %rax
addq %rax, %rdx
addsd %xmm1, %xmm0
movl -12(%rbp), %eax
cltq
movsd %xmm0, (%rdx,%rax,8)
movl -16(%rbp), %eax
cmpl -12(%rbp), %eax
je .L10
movl -16(%rbp), %eax
movslq %eax, %rdx
movq %rdx, %rax
addq %rax, %rax
addq %rdx, %rax
salq $3, %rax
movq %rax, %rdx
movq -40(%rbp), %rax
leaq (%rdx,%rax), %rcx
movl -12(%rbp), %eax
movslq %eax, %rdx
movq %rdx, %rax
addq %rax, %rax
addq %rdx, %rax
salq $3, %rax
movq %rax, %rdx
movq -40(%rbp), %rax
addq %rax, %rdx
movl -12(%rbp), %eax
cltq
movsd (%rcx,%rax,8), %xmm0
movl -16(%rbp), %eax
cltq
movsd %xmm0, (%rdx,%rax,8)
addl $1, -8(%rbp)
movl -8(%rbp), %eax
cmpl -52(%rbp), %eax

```

```

    jl     .L11
    addl $1, -12(%rbp)
    movl -12(%rbp), %eax
    cmpl -16(%rbp), %eax
    jle    .L12
    movl -16(%rbp), %eax
    cltq
    leaq 0(,%rax,8), %rdx
    movq -48(%rbp), %rax
    addq %rdx, %rax
    pxor %xmm0, %xmm0
    movsd %xmm0, (%rax)
    movl $0, -4(%rbp)
    jmp    .L13
    movl -16(%rbp), %eax
    cltq
    leaq 0(,%rax,8), %rdx
    movq -48(%rbp), %rax
    addq %rdx, %rax
    movsd (%rax), %xmm1
    movl -4(%rbp), %eax
    cltq
    leaq 0(,%rax,8), %rdx
    movq -32(%rbp), %rax
    addq %rdx, %rax
    movsd (%rax), %xmm2
    movl -4(%rbp), %eax
    movslq %eax, %rdx
    movq %rdx, %rax
    addq %rax, %rax
    addq %rdx, %rax
    salq $3, %rax
    movq %rax, %rdx
    movq -24(%rbp), %rax
    addq %rax, %rdx
    movl -16(%rbp), %eax
    cltq
    movsd (%rdx,%rax,8), %xmm0
    mulsd %xmm2, %xmm0
    movl -16(%rbp), %eax
    cltq
    leaq 0(,%rax,8), %rdx
    movq -48(%rbp), %rax
    addq %rdx, %rax
    addsd %xmm1, %xmm0
    movsd %xmm0, (%rax)
    addl $1, -4(%rbp)
    movl -4(%rbp), %eax
    cmpl -52(%rbp), %eax
    jl     .L14
    addl $1, -16(%rbp)
    movl -16(%rbp), %eax
    cmpl -56(%rbp), %eax
    jl     .L15
    nop
    popq %rbp
    ret
main:
    pushq %rbp
    movq %rsp, %rbp
    subq $432, %rsp

```

```

movq %fs:40, %rax
movq %rax, -8(%rbp)
xorl %eax, %eax
movl $5, -424(%rbp)
movl $3, -420(%rbp)
movl -420(%rbp), %ecx
movl -424(%rbp), %edx
leaq -384(%rbp), %rsi
leaq -224(%rbp), %rax
movq %rax, %rdi
call get_data
movl -420(%rbp), %r8d
movl -424(%rbp), %edi
leaq -416(%rbp), %rcx
leaq -304(%rbp), %rdx
leaq -384(%rbp), %rsi
leaq -224(%rbp), %rax
movl %r8d, %r9d
movl %edi, %r8d
movq %rax, %rdi
call square
movl $0, %eax
movq -8(%rbp), %rdi
xorq %fs:40, %rdi
je .L18
call_stack_chk_fail@PLT
leave
ret

```