

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. 8304

Мешков М.А.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

Для заданного варианта программы обработки данных (программа 10 - Интегрирование методом Симпсона), представленной на языке Паскаль, разработать вычислительный алгоритм и также варианты программ его реализации на языках программирования Си и Ассемблер.

Ход выполнения.

1. Был выполнен ручной расчет характеристик программы на языке Паскаль — см. табл. 1, табл. 2, табл. 3. Исходный код программы см. в приложении А.

При ручном расчете (а также автоматическом далее) используются $E_{\text{крит}}=3000$, число Страуда = 10, число внешних связей = 4 (т. к. функция `simp` принимает 4 аргумента).

Таблица 1 — Количество операторов и операндов программы на языке Паскаль

№	Оператор	Количество	Операнд	Количество
1	()	17	0.0	2
2	*	9	1	1
3	+	8	1.0	3
4	-	4	1.0E-6	1
5	/	6	2	3
6	<>	1	2.0	2
7	=	18	3.0	2
8	>=	1	4.0	2
9	abs	2	9.0	1
10	and	1	delta_x	7
11	const	1	end_sum	4
12	for	1	even_sum	5
13	fx	5	fx	1
14	program	1	i	2
15	real	1	lower	9
16	repeat	1	odd_sum	8

17	simps	2	pieces	7
18			simp1	1
19			sum	9
20			sum1	4
21			tol	4
22			upper	7
23			x	5

Таблица 2 — Измеримые характеристики программы на языке Паскаль

Число простых операторов	17
Число простых операндов	23
Общее число всех операторов	79
Общее число всех операндов	90
Словарь программы	40
Длина программы	169

Таблица 3 — Расчетные характеристики программы на языке Паскаль

Оценка длины программы	173.53
Реальный объем работы	899.41
Потенциальный объем работы	15.51
Уровень программы	0.0172
Оценка уровня программы	0.03
Интеллектуальное содержание программы	27.04
Работа программиста	52156.2
Время программирования	5215.62
Оценка времени программирования	3071.67
Уровень используемого языка программирования	0.267
Ожидаемое число ошибок в программе	1

2. Был выполнен автоматический расчет характеристик программы на языке Паскаль — см. рис. 1. Результаты совпали с ручным расчетом.

Operators:			Summary:		
1	17	()	=====		
2	9	*	The number of different operators	:	17
3	8	+	The number of different operands	:	23
4	4	-	The total number of operators	:	79
5	6	/	The total number of operands	:	90
6	1	<>			
7	18	=	Dictionary	(D)	: 40
8	1	>=	Length	(N)	: 169
9	2	abs	Length estimation	(^N)	: 173.529
10	1	and	Volume	(V)	: 899.406
11	1	const	Potential volume	(*V)	: 15.5098
12	1	for	Limit volume	(**V)	: 25.8496
13	5	fx	Programming level	(L)	: 0.0172445
14	1	program	Programming level estimation	(^L)	: 0.0300654
15	1	real	Intellect	(I)	: 27.041
16	1	repeat	Time of programming	(T)	: 5215.62
17	2	sims	Time estimation	(^T)	: 3071.67
Operands:			Programming language level	(lambda)	: 0.267458
1	2	0.0	Work on programming	(E)	: 52156.2
2	1	1	Error	(B)	: 0.465317
3	3	1.0	Error estimation	(^B)	: 0.299802
4	1	1.0E-6			
5	3	2			
6	2	2.0			
7	2	3.0			
8	2	4.0			
9	1	9.0			
10	7	delta_x			
11	4	end_sum			
12	5	even_sum			
13	1	fx			
14	2	i			
15	9	lower			
16	8	odd_sum			
17	7	pieces			
18	1	simpl			
19	9	sum			
20	4	sum1			
21	4	tol			
22	7	upper			
23	5	x			

Рисунок 1 - Автоматический расчет характеристик программы на языке Паскаль

3. Был выполнен ручной расчет характеристик программы на языке Си — см. табл. 4, табл. 5, табл. 6. Исходный код программы см. в приложении В.

Таблица 4 - Количество операторов и операндов программы на языке Си

№	Оператор	Количество	Операнд	Количество
1	!=	1	0.0	2
2	&&	1	1	2
3	()	10	1.0	2
4	* (умножение)	8	1.0e-6	1
5	*=	1	2	4
6	+	7	2.0	1
7	++	1	3.0	2
8	+=	1	4.0	2

9	,	11	9.0	1
10	-	4	delta_x	6
11	/	6	end_sum	3
12	<=	1	even_sum	4
13	=	16	i	4
14	>=	1	lower	8
15	& (взятие адреса)	1	odd_sum	6
16	* (разыменование)	6	pieces	5
17	* (часть типа)	1	sum	9
18	do while	1	sum1	4
19	fabs	2	tol	4
20	for	1	upper	6
21	fx	5	x	4
22	main	1		
23	return	1		
24	simps	2		

Таблица 5 — Измеримые характеристики программы на языке Си

Число простых операторов	24
Число простых операндов	21
Общее число всех операторов	90
Общее число всех операндов	80
Словарь программы	45
Длина программы	170

Таблица 6 — Расчетные характеристики программы на языке Си

Оценка длины программы	202.28
Реальный объем работы	933.61
Потенциальный объем работы	15.51
Уровень программы	0.0166
Оценка уровня программы	0.022
Интеллектуальное содержание программы	20.42

Работа программиста	56199.2
Время программирования	5619.92
Оценка времени программирования	5078.31
Уровень используемого языка программирования	0.2576
Ожидаемое число ошибок в программе	1

4. Был выполнен автоматический расчет характеристик программы на языке Си — см. рис. 2. Результаты совпали с ручным расчетом.

Operators:			Summary:		
1	1	!=	=====		
2	1	&&	The number of different operators	:	24
3	10	()	The number of different operands	:	21
4	8	*	The total number of operators	:	90
5	1	*=	The total number of operands	:	80
6	7	+			
7	1	++	Dictionary	(D)	: 45
8	1	+=	Length	(N)	: 170
9	11	,	Length estimation	(^N)	: 202.278
10	4	-	Volume	(V)	: 933.615
11	6	/	Potential volume	(*V)	: 15.5098
12	1	<=	Limit volume	(**V)	: 25.8496
13	16	=	Programming level	(L)	: 0.0166126
14	1	>=	Programming level estimation	(^L)	: 0.021875
15	1	_&	Intellect	(I)	: 20.4228
16	6	_*	Time of programming	(T)	: 5619.92
17	1	__*	Time estimation	(^T)	: 5078.31
18	1	dowhile	Programming language level	(lambda)	: 0.257658
19	2	fabs	Work on programming	(E)	: 56199.2
20	1	for	Error	(B)	: 0.489064
21	5	fx	Error estimation	(^B)	: 0.311205
22	1	main			
23	1	return			
24	2	simps			
Operands:					
1	2	0.0			
2	2	1			
3	2	1.0			
4	1	1.0e-6			
5	4	2			
6	1	2.0			
7	2	3.0			
8	2	4.0			
9	1	9.0			
10	6	delta_x			
11	3	end_sum			
12	4	even_sum			
13	4	i			
14	8	lower			
15	6	odd_sum			
16	5	pieces			
17	9	sum			
18	4	sum1			
19	4	tol			
20	6	upper			
21	4	x			

Рисунок 2 - Автоматический расчет характеристик программы на языке Си

5. Был выполнен ручной расчет характеристик программы на языке Ассемблер — см. табл. 7, табл. 8, табл. 9. Исходный код программы см. в приложении С.

Программа на языке Ассемлера была получена с помощью команды `gcc main.c -S -masm=intel -fno-asynchronous-unwind-tables`. При расчете исключались директивы описаний.

Таблица 7 - Количество операторов и операндов программы на языке Ассемлера

№	Оператор	Количество	Операнд	Количество
1	push	3	-8[rbp]	6
2	mov	25	-72[rbp]	6
3	movsd	42	-80[rbp]	4
4	sub	4	-88[rbp]	2
5	add	3	-96[rbp]	7
6	pxor	5	-56[rbp]	5
7	subsd	3	-32[rbp]	8
8	cvtsi2sd	3	-40[rbp]	5
9	mulsd	6	-48[rbp]	7
10	movapd	3	-24[rbp]	5
11	movq	11	-104[rbp]	2
12	divsd	5	-16[rbp]	6
13	addsd	9	-52[rbp]	4
14	call	6	rbp	7
15	lea	1	.LC0[rip]	2
16	ucomisd	2	.LC5[rip]	1
17	jp	1	.LC6[rip]	1
18	je	2	.LC2[rip]	2
19	leave	2	.LC3[rip]	2
20	nop	1	.LC4[rip]	2
21	xor	1	[rax]	6
22	andpd	2	rax	22
23	comisd	1	-1598689907	1

24	jnb	1	-1	1
25	shr	1	rsp	5
26	sar	1	xmm0	76
27	cmp	1	xmm1	33
28	jle	1	xmm2	4
29	jmp	1	fx	4
30	sal	1	eax	13
31	pop	1	.L5	1
32	.align	6	.L8	1
33	.long	14	.L9	1
34			.L10	1
35			fs:40	2
36			__stack_chk_fail@PLT	1
37			simps	1
38			.L13	1
39			rdi	2
40			104	1
41			2	1
42			1	3
43			.L4	1
44			31	1
45			edx	3
46			.L5	1
47			48	1
48			rdx	4
49			0	7
50			8	5
51			1072693248	1
52			1074790400	1
53			1074266112	1
54			2147483647	1
55			1051772663	1
56			1075970048	1

57			16	1
----	--	--	----	---

Таблица 8 — Измеримые характеристики программы на языке Ассемлера

Число простых операторов	33
Число простых операндов	57
Общее число всех операторов	169
Общее число всех операндов	295
Словарь программы	90
Длина программы	464

Таблица 9 — Расчетные характеристики программы на языке Ассемлера

Оценка длины программы	498.9397367462192
Реальный объем работы	3012.219836696969
Потенциальный объем работы	15.509775004326936
Уровень программы	0.005148951884379755
Оценка уровня программы	0.011710323574730355
Интеллектуальное содержание программы	35.27406896594294
Работа программиста	585016.1167431112
Время программирования	58501.61167431112
Оценка времени программирования	27659.726515733266
Уровень используемого языка программирования	0.0798590852348352
Ожидаемое число ошибок в программе	$\text{ceil}(3.012219836696969) = 4$

6. Была сформирована сводная таблица — табл. 10.

Таблица 10 — Сводная таблица характеристик программ

Характеристика	Паскаль	Си	Ассемблер
Число простых операторов	17	24	33
Число простых операндов	23	21	57
Общее число всех операторов	79	90	169
Общее число всех операндов	90	80	295

Словарь программы	40	45	90
Длина программы	169	170	464
Оценка длины программы	173.53	202.28	498.9397
Реальный объем работы	899.41	933.61	3012.2198
Потенциальный объем работы	15.51	15.51	15.5097
Уровень программы	0.0172	0.0166	0.0051
Оценка уровня программы	0.03	0.022	0.0117
Интеллектуальное содержание программы	27.04	20.42	35.2740
Работа программиста	52156.2	56199.2	585016.1167
Время программирования	5215.62	5619.92	58501.6116
Оценка времени программирования	3071.67	5078.31	27659.7265
Уровень используемого языка программирования	0.267	0.2576	0.0798
Ожидаемое число ошибок в программе	1	1	4

Выводы.

Метрические характеристики программ на Си и Паскаль схожи. Однако характеристики программы на Ассемблере сильно отличаются, связано это с тем что ассемблер — язык более низкого уровня.

ПРИЛОЖЕНИЕ А. ПРОГРАММА НА ЯЗЫКЕ ПАСКАЛЬ

```
program simp1;
{ integration by Simpson's method }
{ Turbo Pascal cannot pass function names as arguments }

const tol = 1.0E-6;
var sum,upper,lower: real;

function fx(x: real): real;
{ find f(x)=1/x }
{ watch out for x=0 }
begin
    fx:=1.0/x
end; { function fx }

procedure simps(
    lower,upper,tol: real;
    var sum: real);
{ numerical integration by Simpson's rule }
{ function is fx, limits are lower and upper }
{ with number of regions equal to pieces }
{ partition is delta_x, answer is sum }
var i: integer;
    x,delta_x,even_sum,
    odd_sum,end_sum,
    sum1: real;
    pieces: integer;
begin
    pieces:=2;
    delta_x:=(upper-lower)/pieces;
    odd_sum:=fx(lower+delta_x);
    even_sum:=0.0;
    end_sum:=fx(lower)+fx(upper);
    sum:=(end_sum+4.0*odd_sum)*delta_x/3.0;
    {writeln(pieces:5,sum);}
    repeat
        pieces:=pieces*2;
        sum1:=sum;
```

```

    delta_x:=(upper-lower)/pieces;
    even_sum:=even_sum+odd_sum;
    odd_sum:=0.0;
    for i:=1 to pieces div 2 do
        begin
            x:=lower+delta_x*(2.0*i-1.0);
            odd_sum:=odd_sum+fx(x)
        end;
    sum:=(end_sum+4.0*odd_sum+2.0*even_sum)*delta_x/3.0;
    {writeln(pieces:5,sum)}
    until (sum<>sum1) and (abs(sum-sum1)>=abs(tol*sum))
end; { simps }

begin { main program }
    {ClrScr;}
    lower:=1.0;
    upper:=9.0;
    {writeln;}
    simps(lower,upper,tol,sum);
    {writeln;}
    {writeln(chr(7),'area= ',sum)}
end.

```

ПРИЛОЖЕНИЕ В. ПРОГРАММА НА ЯЗЫКЕ СИ

```
// #include <stdio.h>
#include <math.h>

double fx(double x) {
    return 1.0 / x;
}

void simps(double lower, double upper, double tol, double *sum) {
    int pieces = 2;
    double
        delta_x = (upper-lower)/pieces,
        odd_sum = fx(lower+delta_x),
        even_sum = 0.0,
        end_sum = fx(lower) + fx(upper);
    *sum = (end_sum + 4.0 * odd_sum)*delta_x/3.0;
    // printf("%5d %f\n", pieces, *sum);
    double sum1;
    do {
        pieces *= 2;
        sum1 = *sum;
        delta_x = (upper-lower)/pieces;
        even_sum = even_sum + odd_sum;
        odd_sum = 0.0;
        for (int i = 1; i <= pieces/2; i++) {
            double x = lower+delta_x*(2*i-1);
            odd_sum += fx(x);
        }
        *sum = (end_sum + 4.0*odd_sum + 2.0*even_sum)*delta_x/3.0;
```

```
        // printf("%5d %f\n", pieces, *sum);  
    } while (*sum != sum1 && fabs(*sum-sum1) >= fabs(tol**sum));  
}
```

```
int main() {  
    const double tol = 1.0e-6;  
    double  
        lower = 1.0,  
        upper = 9.0,  
        sum;  
    simps(lower, upper, tol, &sum);  
    // printf("\narea=%f\n", sum);  
}
```

ПРИЛОЖЕНИЕ С. ПРОГРАММА НА ЯЗЫКЕ АССЕМБЛЕРА

```
.file "main.c"
```

```
.intel_syntax noprefix
```

```
.text
```

```
.globl fx
```

```
.type fx, @function
```

fx:

```
push rbp
```

```
mov rbp, rsp
```

```
movsd QWORD PTR -8[rbp], xmm0
```

```
movsd xmm0, QWORD PTR .LC0[rip]
```

```
divsd xmm0, QWORD PTR -8[rbp]
```

```
movq rax, xmm0
```

```
movq xmm0, rax
```

```
pop rbp
```

```
ret
```

```
.size fx, .-fx
```

```
.globl simps
```

```
.type simps, @function
```

simps:

```
push rbp
```

```
mov rbp, rsp
```

```
sub rsp, 104
```

```
movsd QWORD PTR -72[rbp], xmm0
```

```
movsd QWORD PTR -80[rbp], xmm1
```

```
movsd QWORD PTR -88[rbp], xmm2
```

```
mov QWORD PTR -96[rbp], rdi
```

```
mov DWORD PTR -56[rbp], 2
```

```
movsd xmm0, QWORD PTR -80[rbp]
```

```
subsd xmm0, QWORD PTR -72[rbp]
```

```

pxor  xmm1, xmm1
cvtsi2sd    xmm1, DWORD PTR -56[rbp]
divsd  xmm0, xmm1
movsd     QWORD PTR -32[rbp], xmm0
movsd     xmm0, QWORD PTR -72[rbp]
addsd  xmm0, QWORD PTR -32[rbp]
movq  rax, xmm0
movq  xmm0, rax
call  fx
movq  rax, xmm0
mov  QWORD PTR -48[rbp], rax
pxor  xmm0, xmm0
movsd     QWORD PTR -40[rbp], xmm0
mov  rax, QWORD PTR -72[rbp]
movq  xmm0, rax
call  fx
movsd     QWORD PTR -104[rbp], xmm0
mov  rax, QWORD PTR -80[rbp]
movq  xmm0, rax
call  fx
addsd  xmm0, QWORD PTR -104[rbp]
movsd     QWORD PTR -24[rbp], xmm0
movsd     xmm1, QWORD PTR -48[rbp]
movsd     xmm0, QWORD PTR .LC2[rip]
mulsd  xmm0, xmm1
addsd  xmm0, QWORD PTR -24[rbp]
mulsd  xmm0, QWORD PTR -32[rbp]
movsd     xmm1, QWORD PTR .LC3[rip]
divsd  xmm0, xmm1
mov  rax, QWORD PTR -96[rbp]

```



```
movsd    QWORD PTR [rax], xmm0
```

.L8:

```
sal      DWORD PTR -56[rbp]
mov  rax, QWORD PTR -96[rbp]
movsd    xmm0, QWORD PTR [rax]
movsd    QWORD PTR -16[rbp], xmm0
movsd    xmm0, QWORD PTR -80[rbp]
subsd    xmm0, QWORD PTR -72[rbp]
pxor     xmm1, xmm1
cvtsi2sd  xmm1, DWORD PTR -56[rbp]
divsd    xmm0, xmm1
movsd    QWORD PTR -32[rbp], xmm0
movsd    xmm0, QWORD PTR -40[rbp]
addsd    xmm0, QWORD PTR -48[rbp]
movsd    QWORD PTR -40[rbp], xmm0
pxor     xmm0, xmm0
movsd    QWORD PTR -48[rbp], xmm0
mov  DWORD PTR -52[rbp], 1
jmp  .L4
```

.L5:

```
mov  eax, DWORD PTR -52[rbp]
add  eax, eax
sub  eax, 1
pxor  xmm0, xmm0
cvtsi2sd  xmm0, eax
mulsd  xmm0, QWORD PTR -32[rbp]
movsd  xmm1, QWORD PTR -72[rbp]
addsd  xmm0, xmm1
movsd  QWORD PTR -8[rbp], xmm0
mov  rax, QWORD PTR -8[rbp]
```

```

movq xmm0, rax
call  fx
movsd    xmm1, QWORD PTR -48[rbp]
addsd xmm0, xmm1
movsd    QWORD PTR -48[rbp], xmm0
add  DWORD PTR -52[rbp], 1

```

.L4:

```

mov  eax, DWORD PTR -56[rbp]
mov  edx, eax
shr  edx, 31
add  eax, edx
sar  eax
cmp  DWORD PTR -52[rbp], eax
jle  .L5
movsd    xmm1, QWORD PTR -48[rbp]
movsd    xmm0, QWORD PTR .LC2[rip]
mulsd xmm0, xmm1
movapd    xmm1, xmm0
addsd xmm1, QWORD PTR -24[rbp]
movsd    xmm0, QWORD PTR -40[rbp]
addsd xmm0, xmm0
addsd xmm0, xmm1
mulsd xmm0, QWORD PTR -32[rbp]
movsd    xmm1, QWORD PTR .LC3[rip]
divsd xmm0, xmm1
mov  rax, QWORD PTR -96[rbp]
movsd    QWORD PTR [rax], xmm0
mov  rax, QWORD PTR -96[rbp]
movsd    xmm0, QWORD PTR [rax]
ucomisd  xmm0, QWORD PTR -16[rbp]

```

```

jp    .L9
ucomisd    xmm0, QWORD PTR -16[rbp]
je    .L10

```

.L9:

```

mov    rax, QWORD PTR -96[rbp]
movsd   xmm0, QWORD PTR [rax]
subsd  xmm0, QWORD PTR -16[rbp]
movq   xmm1, QWORD PTR .LC4[rip]
andpd  xmm0, xmm1
mov    rax, QWORD PTR -96[rbp]
movsd   xmm1, QWORD PTR [rax]
mulsd  xmm1, QWORD PTR -88[rbp]
movq   xmm2, QWORD PTR .LC4[rip]
andpd  xmm1, xmm2
comisd  xmm0, xmm1
jnb    .L8

```

.L10:

```

nop
leave
ret
.size   simps, .-simps
.globl  main
.type   main, @function

```

main:

```

push    rbp
mov     rbp, rsp
sub     rsp, 48
mov     rax, QWORD PTR fs:40
mov     QWORD PTR -8[rbp], rax
xor     eax, eax

```

```

movsd    xmm0, QWORD PTR .LC5[rip]
movsd    QWORD PTR -32[rbp], xmm0
movsd    xmm0, QWORD PTR .LC0[rip]
movsd    QWORD PTR -24[rbp], xmm0
movsd    xmm0, QWORD PTR .LC6[rip]
movsd    QWORD PTR -16[rbp], xmm0
lea      rdx, -40[rbp]
movsd    xmm1, QWORD PTR -32[rbp]
movsd    xmm0, QWORD PTR -16[rbp]
mov      rax, QWORD PTR -24[rbp]
mov      rdi, rdx
movapd   xmm2, xmm1
movapd   xmm1, xmm0
movq     xmm0, rax
call     simps
mov      eax, 0
mov      rdx, QWORD PTR -8[rbp]
sub      rdx, QWORD PTR fs:40
je       .L13
call     __stack_chk_fail@PLT

```

.L13:

```

leave
ret
.size   main, .-main
.section .rodata
.align 8

```

.LC0:

```

.long 0
.long 1072693248
.align 8

```

```
.LC2:
    .long 0
    .long 1074790400
    .align 8
.LC3:
    .long 0
    .long 1074266112
    .align 16
.LC4:
    .long -1
    .long 2147483647
    .long 0
    .long 0
    .align 8
.LC5:
    .long -1598689907
    .long 1051772663
    .align 8
.LC6:
    .long 0
    .long 1075970048
    .ident "GCC: (GNU) 11.1.0"
    .section      .note.GNU-stack,"",@progbits
```