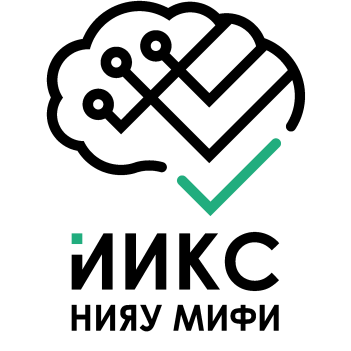
**Национальный исследовательский ядерный университет «МИФИ»**

**Институт интеллектуальных кибернетических систем**

**Кафедра №12 «Компьютерные системы и технологии»**



**ОТЧЕТ**

**О выполнении лабораторной работы №2**

**«Работа с вещественными числами.**

**Обеспечение заданной точности**

**вычислений»**

Студент: Кафанов С.П.

Группа: Б21-515

Преподаватель: Храпов А.С.

Москва — 2021

**Задача. Вариант №5**

Написать алгоритм, вычисляющий сумму ряда y используя два способа задания точности вычисления: через Ꜫ или задание количества итераций n.

**Использованные типы данных.**

Long double для работы с нецелочисленными данными с наибольшей возможной точностью и int для работы с целыми числами.

**Код.**

1 #include <stdio.h>

2

3 long double f1(long double x, long double eps);

4 long double f2(long double x, int n);

5

6 int main() {

7 long double x = 0;

8 int n = 0;

9 long double eps = 0;

10

11 printf("Enter x: ");

12 if (scanf("%Lf", &x) <= 0) { printf("Error!!!"); }

13 printf("Enter epsilon: ");

14 if (scanf("%Lf", &eps) <= 0) { printf("Error!!!"); }

15 printf("Enter n: ");

16 if (scanf("%d", &n) <= 0) { printf("Error!!!"); }

17

18 printf("%.30Lf\n", f1(x, eps));

19 printf("%.30Lf\n", f2(x, n));

20 return 0;

21 }

22

23 long double f1(long double x, long double eps) {

24 long double res = 0, s1 = 0, p1 = 1, p2 = 0;

25 int k = 2;

26 p1 = p1\*x\*x\*2;

27 res += p1;

28 do {

29 //printf("%.10Lf %.10Lf %.10Lf ", p1, p2, res);

30 p2 = p1;

31 s1 = res;

32 p1 \*= x\*x\*2/k;

33 //printf("%.20Lf %.20Lf\n", p1, p2-p1);

34 res += p1;

35 k++;

36 } while (p1 > eps);

37 return res;

38 }

39

40 long double f2(long double x, int n) {

41 long double res = 0, p1 = 1, p2 = 0;

42 p1 = p1\*x\*x\*2;

43 int k = 2;

44 res += p1;

45 do {

46 p2 = p1;

47 p1 = p1 \* x \* x \* 2 / k;

48 res += p1;

49 k++;

50 } while (n >= k);

51 return res;

52 }

**Тесты.**

Проверка точности вычислений производилась через сайт <https://www.wolframalpha.com>.

1) x = 3:

Ꜫ=0.000000000001: 65659968.1373305111/4468253218

n=1000000000: 65659968.13733051113/740657456

wolframalpha, n=1000000000: y = 65659968,13733051113878650325906003356921

точность в 18/19 знаков

2) x = -3:

Ꜫ=0.000000000001: 65659968.1373305111/4468253218

n=1000000000: 65659968.13733051113/740657456

wolframalpha, n=1000000000: y = 65659968,13733051113878650325906003356921

точность в 18-19 знаков

3) x = 0,5:

Ꜫ=0.000000000001: 0.6487212707001281/22769622524002

n=1000000000: 0.648721270700128146/784700644556

wolframalpha, n=1000000000: y = 0.6487212707001281468486507878141635716537761007101480115750793116

точность в 17-19 знаков

4) x = -0,5:

Ꜫ=0.000000000001: 0.6487212707001281/22769622524002

n=1000000000: 0.648721270700128146/784700644556

wolframalpha, n=1000000000: y = 0.6487212707001281468486507878141635716537761007101480115750793116

точность в 17-19 знаков

5) x = 1:

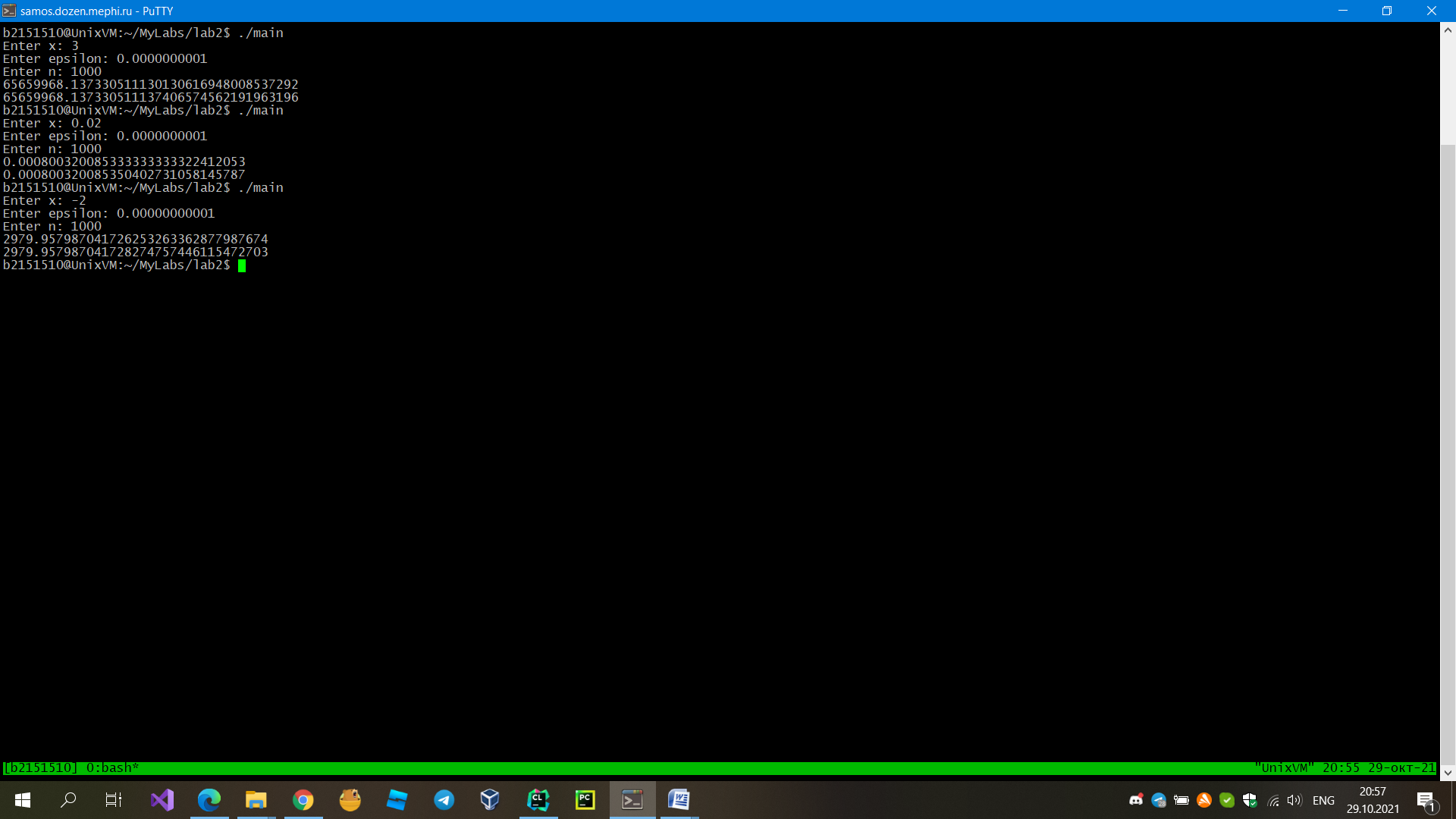
Ꜫ=0.000000000001: 6.3890560989306/49873948034445093

n=1000000000: 6.389056098930650227/397942675367

wolframalpha, n=1000000000: y = 6.3890560989306502272304274605750078131803155705518473240871278225

точность в 14-19 знаков

**Скриншоты.**



**Блок схема функций.**

Возврат: res

Возврат: res

Аргументы: x, eps

Аргументы: x, n

p1<eps

n>=k

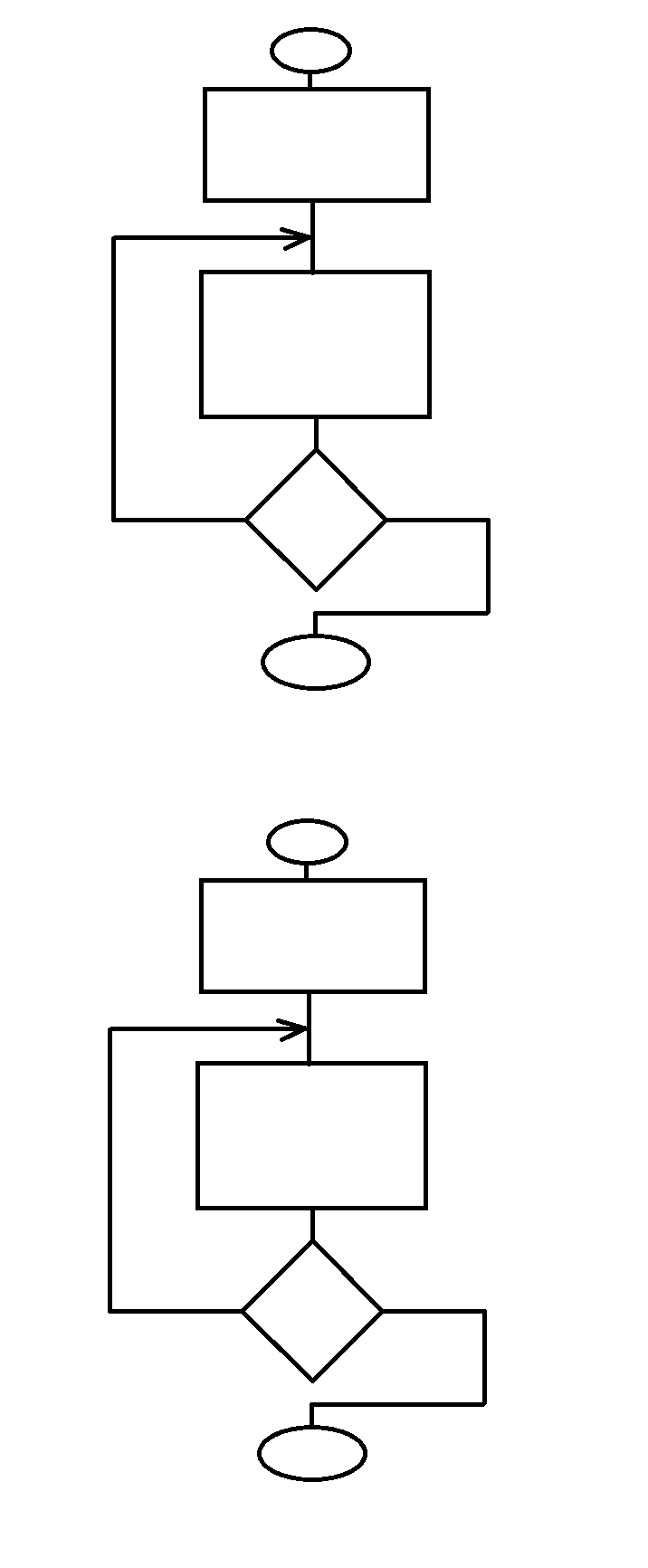
p2 = p1;  
p1= p1\*x\*x\*2/k;  
res += p1;  
k++;

p2 = p1;  
p1= p1\*x\*x\*2/k;  
res += p1;  
k++;

res = 0, p1 = 1;

p2 = 0;  
p1 = p1\*x\*x\*2;  
k = 2;  
res = res + p1;

res = 0, p1 = 1, p2 = 0;  
p1 = p1\*x\*x\*2;  
k = 2;  
res = res + p1;



**Выводы:**

Я получил начальный опыт работы с вещественными числами в ЯП Си. Из интересных замечаний по лабораторной работе: точность вычислений оказывается больше на несколько порядков при работе с функцией, принимающей на вход значение кол-ва операций n.