



**«Московский государственный технический университет
имени Н.Э. Баумана»
(национальный исследовательский университет)
(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ
КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)


Отчет

по домашнему заданию № 1

**Название домашнего задания: Вычисления. Погрешности вычислений.
Программирование разветвляющегося вычислительного процесса.
Программирование циклического процесса. Типы циклов**

Дисциплина: Алгоритмизация и программирование

Студент гр. ИУ6-14Б

 **30.09.2023**

(Подпись, дата)

А.С. Воеводин

(И.О. Фамилия)

Преподаватель

 **30.09.2023**

(Подпись, дата)

О.А. Веселовская

(И.О. Фамилия)

Цель задания – Познакомиться с вычислениями чисел с плавающей запятой в программе, с абсолютной, предельной абсолютной, относительной, предельной относительной погрешностями, их вычислением, видами погрешностей.

Задание – Вычисление тригонометрических, гиперболических функций, длины кривой для чисел с плавающей запятой с помощью циклов и ветвлений.

Ход работы:

- Вывод результата для первой части первого пункта
- Выполнение оценки абсолютной и относительной погрешности представления числа 1 и вычислений над числами типа float. Описание типов погрешностей
- Вычисление левой части в формуле $\text{ch}^2 x - \text{sh}^2 x = 1$ и следование инструкциям
- Разработка программы, которая проверяет равенство $\sin^2 x + \cos^2 x = 1$. Удостоверение, что погрешность достаточно мала. Пояснение полученного результата
- Написание программы, которая выводит максимум из введенных действительных чисел в виде $\max(X^2, Y^2+5, W^2-5)$
- Вычисление длины кривой линии
- Вывод.

Для начала создадим новый проект **dz_1** и перепишем код, данный в файле, получим:

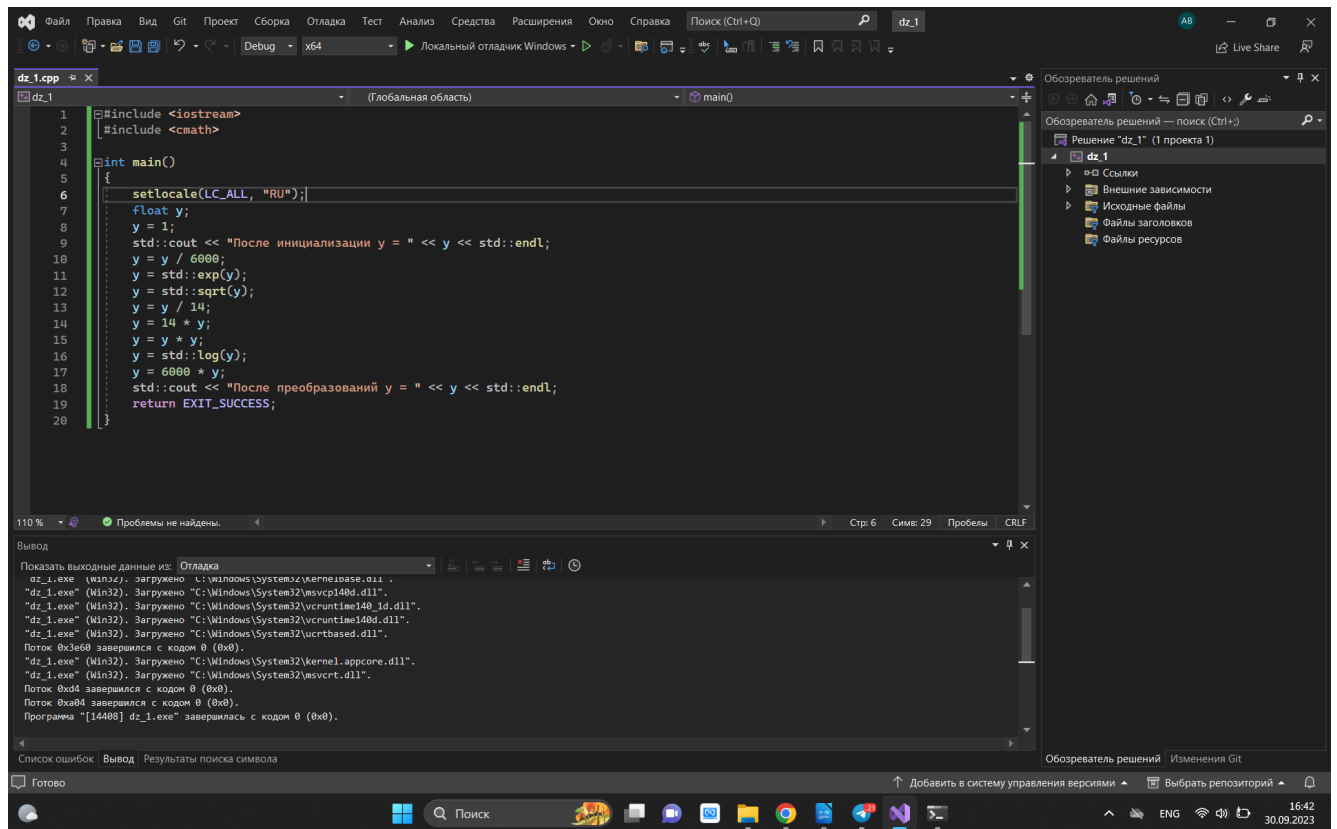


Рисунок 1 – Код программы, данный в файле

После инициализации $y = 1$
После преобразований $y = 0.999844$

Рисунок 2 – Вывод

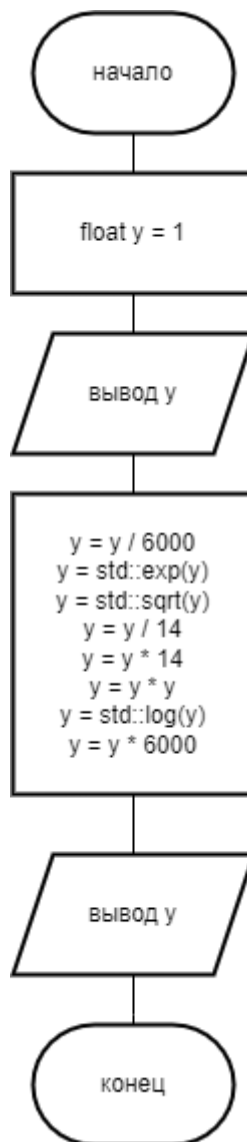


Рисунок 3 – Схема алгоритма

Как мы видим, значение y не совпадает с начальным. Это произошло из-за погрешностей. В нашем случае абсолютная погрешность равна

$\Delta = |1 - 0.999844|$, $\Delta = 0.000156$, а относительная погрешность равна

$\delta = \frac{\Delta}{1} = \Delta = 0.000156$. В данной программе мы можем наблюдать такие виды погрешностей как:

- Погрешность метода (так как функции корня и логарифма вычисляются неточно)
- Начальная погрешность (так как экспонента вычисляется до определённого знака)
- Погрешность округления (так как `float` ограничен 4 байтами)
- Погрешность операций (так как выполняется умножение `float` на `float`)

Далее необходимо вычислить левую часть гиперболического уравнения вида $\text{ch}^2 x - \text{sh}^2 x = 1$. Чтобы это сделать, воспользуемся разложением гиперболических функций в ряды Маклорена:

$$\text{sh} x = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \dots + \frac{x^{2m-1}}{(2m-1)!} + \dots, \quad x \in R;$$

$$\text{ch} x = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \dots + \frac{x^{2m}}{(2m)!} + \dots, \quad x \in R;$$

Рисунок 4 – разложение гиперболических функций в ряды

Немного упростим операции, тогда получим рекуррентную формулу для

нахождения n-ого члена гиперболического синуса: $R_n = R_{n-1} * \frac{x^2}{(k_n + 1) * (k_n + 2)}$,

где $k_n = 2 * n - 1$. Для гиперболического косинуса получаем:

$R_n = R_{n-1} * \frac{x^2}{(k_n + 1) * (k_n + 2)}$, где $k_n = 2 * n$. Напишем код программы:

```

#include <iostream>
#include <cmath>
#include <iomanip>
#include <stdlib.h>

float sh(float x_)
{
    float summ = x_, prev = x_, eps = 1e-15;

    for (size_t i = 1; std::fabsl(prev) >= eps; i += 2)
    {
        prev *= (x_ * x_) / static_cast<float>((i + 1) * (i + 2));
        summ += prev;
    }

    return summ;
}

float ch(float x_)
{
    float summ = 1, prev = 1, eps = 1e-15;

    for (size_t i = 0; std::fabsl(prev) >= eps; i += 2)
    {
        prev *= (x_ * x_) / static_cast<float>((i + 1) * (i + 2));
        summ += prev;
    }

    return summ;
}

```

Рисунок 5 – Функции для вычисления гиперболических функций

```

int main()
{
    setlocale(LC_ALL, "RU");
    // part 1.1
    /* ... */

    // part 1.2
    float y1 = 0, y2 = 0, y = 0, x = 0, delta_rel = 0, delta_abs = 0, ep = 0;
    std::cout << std::setprecision(16);

    for (float x = 5; x <= 25; x += 5)
    {
        y1 = sh(x);
        y2 = ch(x);
        y = std::powf(y2, 2) - std::powf(y1, 2);
        delta_abs = std::fabs(1 - y);
        delta_rel = delta_abs / 1;
        std::cout << x << std::setw(20) << y1 << std::setw(20) << y2 << std::setw(20) << y << std::setw(20)
            << delta_abs << std::setw(20) << delta_rel << std::endl;
    }

    return EXIT_SUCCESS;
}

```

Рисунок 6 – Функция main

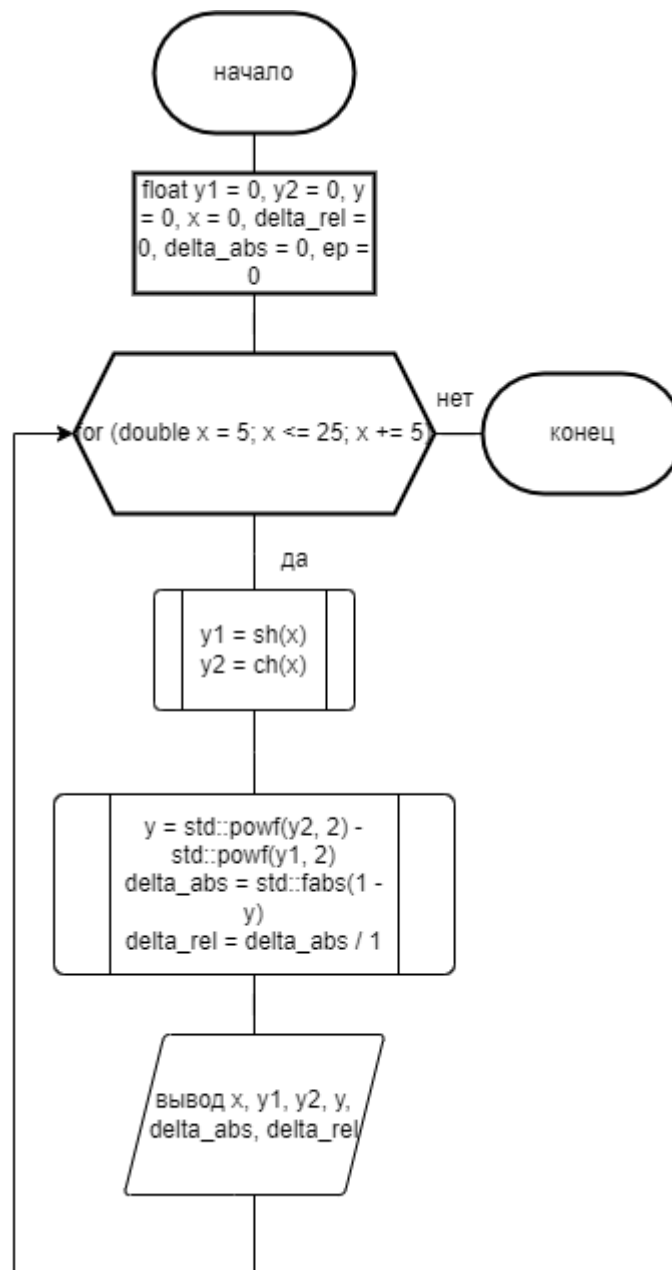


Рисунок 7 – Схема алгоритма

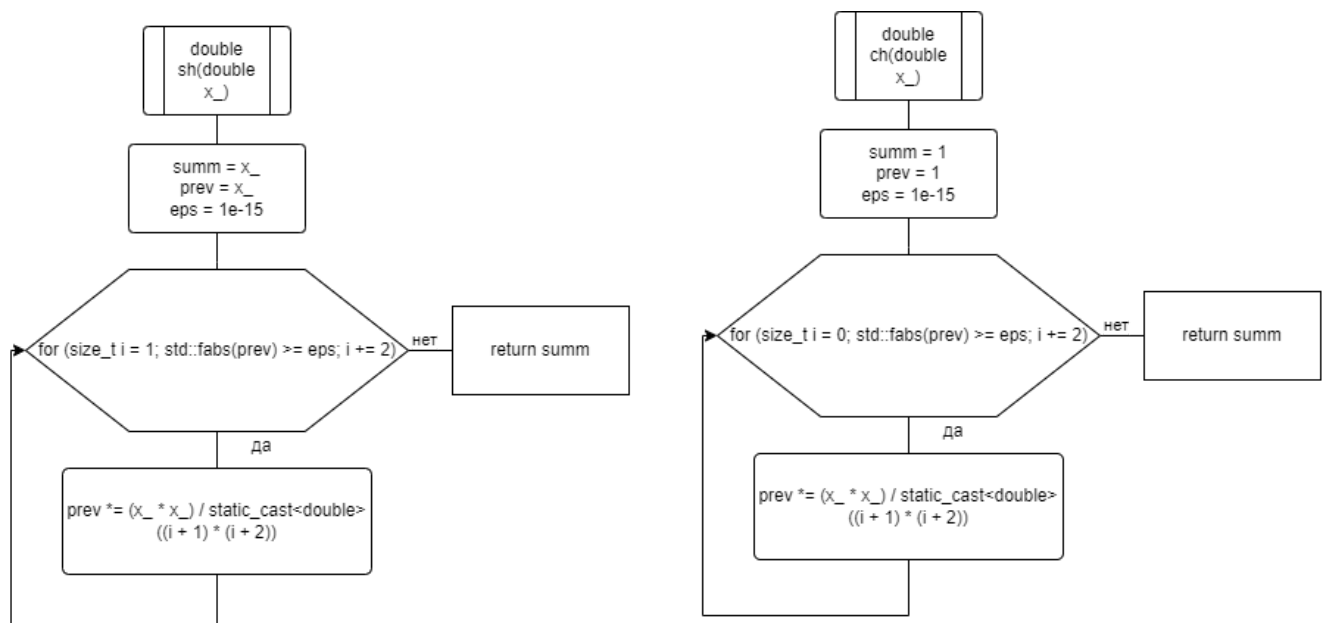


Рисунок 8 – Схема алгоритма гиперболических функций

5	74.20319366455078	74.20994567871094	1.00244140625	0.00244140625	0.00244140625
10	11013.2314453125	11013.2314453125	0	1	1
15	1634508.625	1634508.75	262144	262143	262143
20	242582544	242582592	21474836480	21474836480	21474836480
25	36002451456	36002447360	-281474976710656	281474976710656	281474976710656

Рисунок 9 – Вывод значений float переменных согласно таблице в файле

Наблюдается закономерность: чем больше x , тем выше погрешность, а следовательно – неточность. Тип float представлен в разрядной сетке размером 4 байта, что не позволяет точнее вычислить значение. Заменим float на double и выведем значения:

5	74.20321057778877	74.20994852478785	0.999999999999985	9.094947017729282e-13	9.094947017729282e-13
10	11013.2328747034	11013.23292010332	0.9999998360872269	1.639127731323242e-07	1.639127731323242e-07
15	1634508.686235902	1634508.686236208	1.00244140625	0.00244140625	0.00244140625
20	242582597.7048952	242582597.7048952	-32	33	33
25	36002449668.69292	36002449668.69293	524288	524287	524287

Рисунок 10 – Вывод значений double переменных согласно таблице в файле

Очевидно, что double лучше и точнее, ввиду того, что он представлен в разрядной сетке размером 8 байт.

Изменение любой из переменных повлияет на точность результата:

- Изменение переменной x – повлияет тогда, когда на вход поступит число, не помещающееся в размерную сетку float. Аналогично с y , y_1 , y_2 , так как всегда может быть получено число, не помещающееся в разрядную сетку float. К тому же, каждая из переменных участвует в вычислениях, поэтому любое незначительное округление приведёт к неточности.

Далее напишем программу, проверяющее равенство $\sin^2 x + \cos^2 x = 1$, но сперва поймем как вычислять синус и косинус. В этом нам поможет их разложение в ряды:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} \dots + (-1)^{n-1} \frac{x^{2n-1}}{(2n-1)!} + o(x^{2n})$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} \dots + (-1)^n \frac{x^{2n}}{(2n)!} + o(x^{2n+1})$$

Рисунок 11 – Разложение синуса и косинуса в ряды

Заметим, что это то же самое, что и было в предыдущей программе, за исключением чередования знаков. Напишем программу:

```

double sinus(double x_)
{
    double summ = x_, prev = x_, eps = 1e-15;

    for (size_t i = 1; std::fabsl(prev) >= eps; i += 2)
    {
        prev *= -(x_ * x_) / static_cast<double>((i + 1) * (i + 2));
        summ += prev;
    }

    return summ;
}

double cosinus(double x_)
{
    double summ = 1, prev = 1, eps = 1e-15;

    for (size_t i = 0; std::fabsl(prev) >= eps; i += 2)
    {
        prev *= -(x_ * x_) / static_cast<double>((i + 1) * (i + 2));
        summ += prev;
    }

    return summ;
}

```

Рисунок 12 – Функции синуса и косинуса

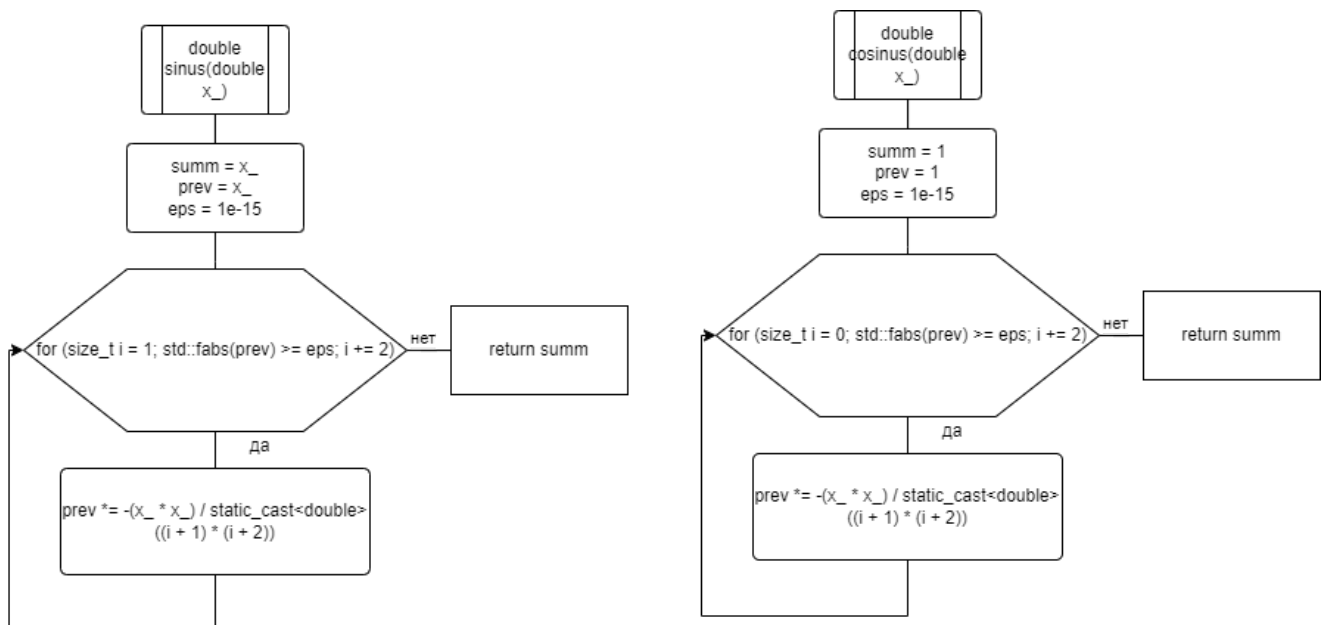


Рисунок 13 – Схема алгоритма тригонометрических функций

```

int main()
{
    setlocale(LC_ALL, "RU");
    // part 1.1
    /* ... */

    // part 1.2
    /* ... */

    // part 1.3
    std::cout << std::setprecision(20);

    for (double i = -1; i <= 1; i += 0.1)
    {
        std::cout << std::fabsl(std::powl(sinus(i), 2) + std::powl(cosinus(i), 2)) << std::endl;
    }

    return EXIT_SUCCESS;
}

```

Рисунок 14 – Функция main

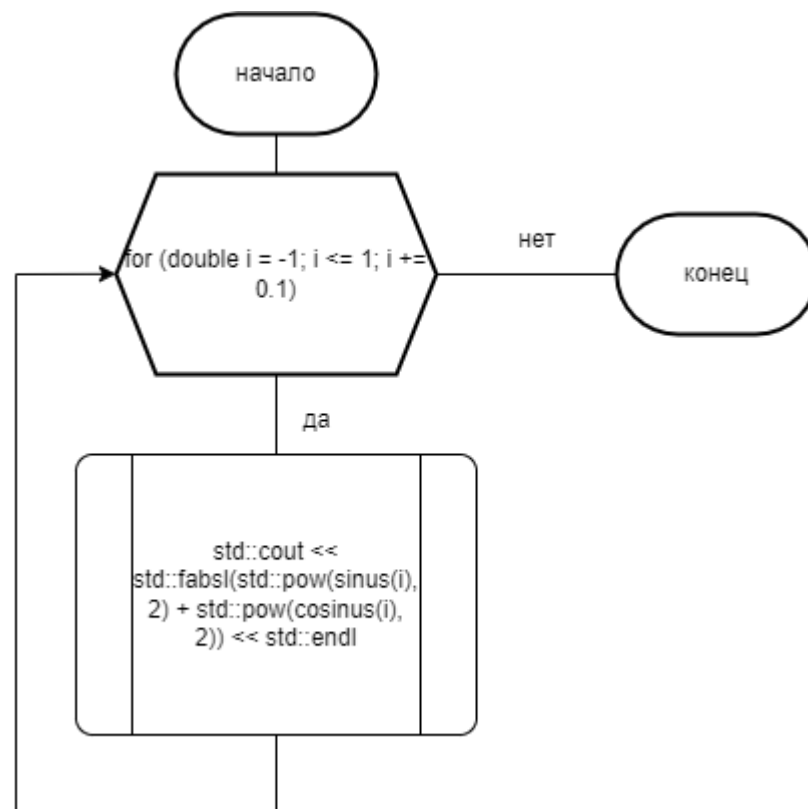


Рисунок 15 – Схема алгоритма

Запустим программу:

```

0.99999999999999988898
1
1
0.99999999999999988898
1
1
0.99999999999999988898
1
1
1.0000000000000000222
1
1
1
1
1
0.99999999999999988898
0.99999999999999988898
1
1.0000000000000000441
1.0000000000000000222
1

```

Рисунок 16 – Вывод значений

Так как double в Visual Studio содержит максимум 15 знаков после запятой, а погрешности начинаются после 15 знака, то можно выводить не 20 цифр после запятой, а 15 – максимальное число в разрядной сетке, тогда получим:

```

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
C:\Users\artem\source\repos\dz_1\x64\Debug\dz_1.
Нажмите любую клавишу, чтобы закрыть это окно:|

```

Рисунок 17 – Вывод значений

Итого получаем полное полное равенство основного тригонометрического уравнения с точностью $\varepsilon = 10^{-16}$.

Следующая задача состоит в том, чтобы вывести максимальное из трёх действительных чисел. Напишем программу:

```
double x = 0, y = 0, w = 0;
std::cout << "Введите x y w: ";
std::cin >> x >> y >> w;

x *= x;
y = y * y + 5;
w = w * w - 5;

if (x >= y && x >= w)
    std::cout << x;
else if (y >= x && y >= w)
    std::cout << y;
else
    std::cout << w;
```

Рисунок 18 – Код программы

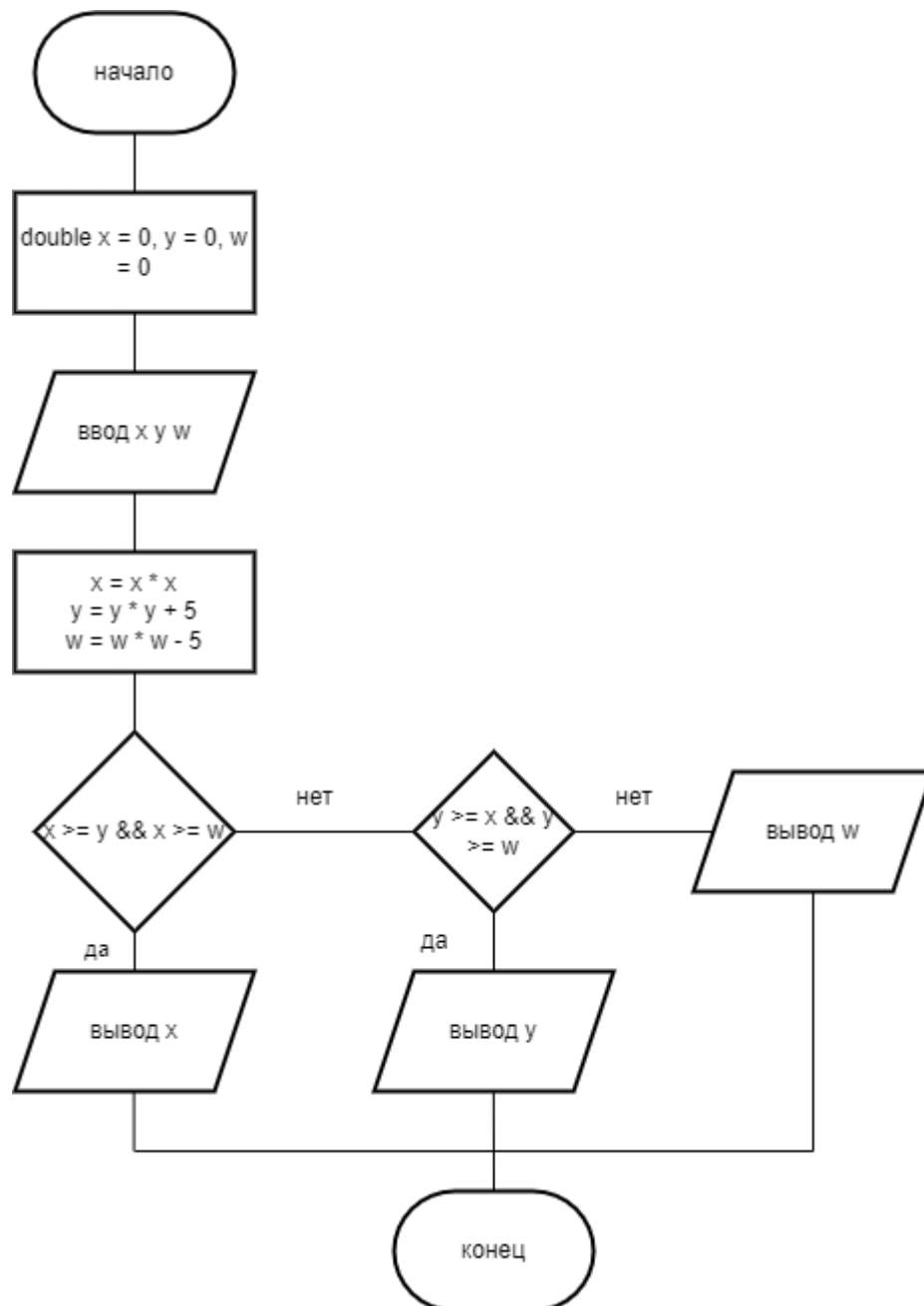


Рисунок 19 – Схема алгоритма

Введём случайные числа:

Введите x y w: 2.7 3.14 1.57
14.8596

Рисунок 20 – Тестовые данные

Введите x y w: 3.23432 6.23423 9.2323
80.2354

Рисунок 21 – Тестовые данные

Введите x y w: 100.1231 43.233 42.0909
10024.6

Рисунок 22 – Тестовые данные

Можем убедиться, что алгоритм работает верно, так как тестовые данные были такими, что алгоритм проверил каждое условие.

Далее необходимо вычислить длину кривой $y = x^{\frac{3}{2}}$ на участке $x \in [0; 4]$.

Длина кривой определяется как длина ломаной линии, которая получается при соединении значений функции в точках разбиения участка на n частей, где длина каждой части – гипотенуза прямоугольного треугольника с катетами Δy , Δx , где $\Delta y = f(x) - f(x - \frac{4}{n})$, $\Delta x = \frac{4}{n}$. Так будет выглядеть программа:

```
double len = 0, eps = 0, correct = 9.073415289388, delta_y = 0, delta_x_2 = 0;
double n = 0.5;
std::cout << std::setprecision(12) << std::fixed;
std::cout << "Введите желаемую точность: ";
std::cin >> eps;

while (std::fabs(len - correct) >= eps)
{
    n *= 2;
    len = 0;
    delta_x_2 = std::pow(4 / n, 2);
    for (double x = 4 / n; x <= 4; x += 4 / n)
    {
        delta_y = f(x) - f(x - 4 / n);
        len += std::sqrt(std::pow(delta_y, 2) + delta_x_2);
    }
}

std::cout << len << ' ' << correct - len << ' ' << n << std::endl;
```

Рисунок 23 – Код программы

```
double f(double x_)
{
    return std::pow(x_, 1.5);
}
```

Рисунок 24 – Код функции

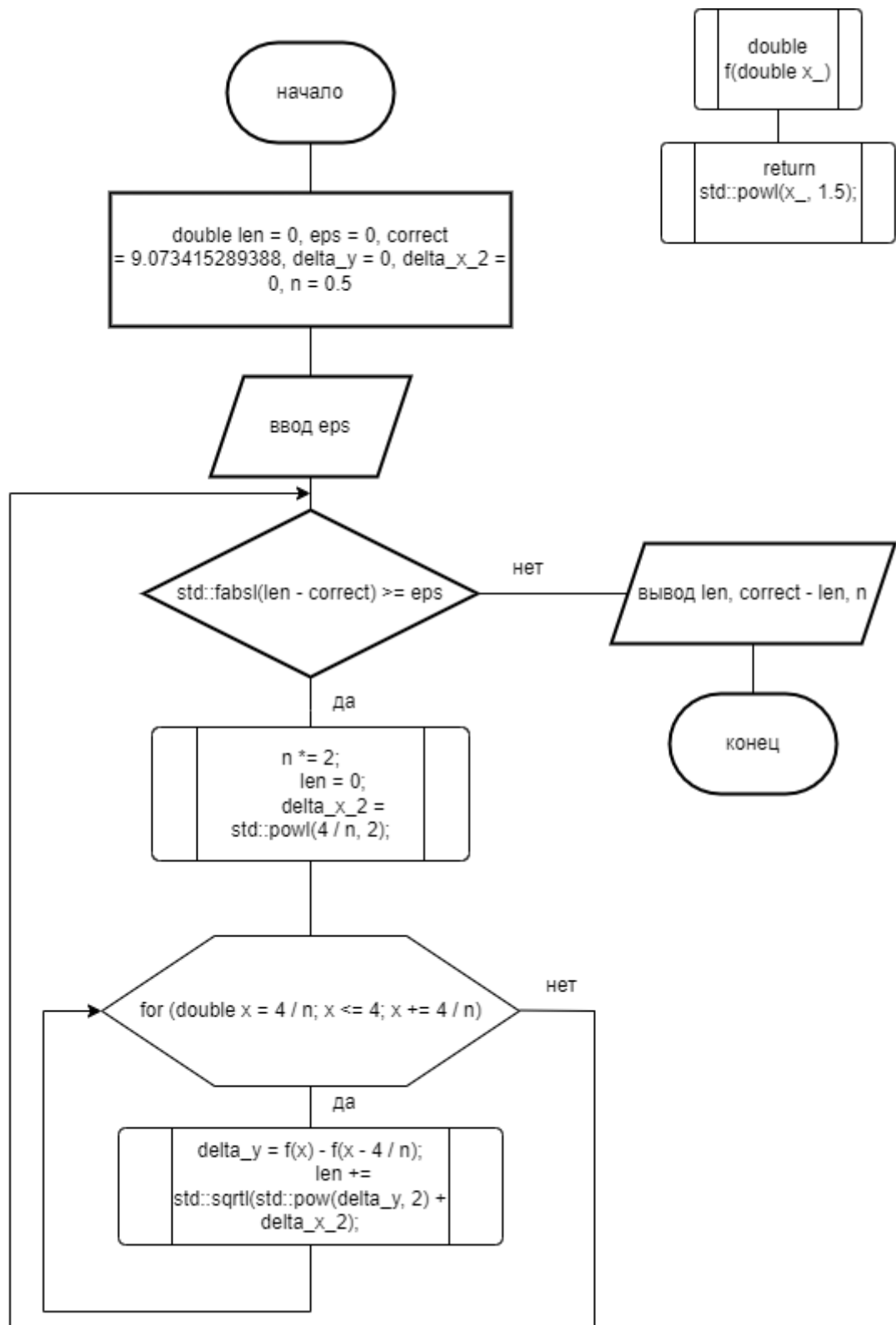


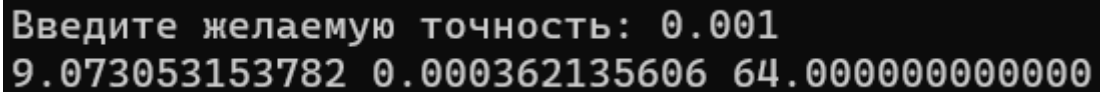
Рисунок 25 – Схема алгоритма

Введём заданную точность:

```

Введите желаемую точность: 0.01
9.069698733270 0.003716556118 16.000000000000
  
```

Рисунок 26 – Проверка программы с первой точностью из файла



```
Введите желаемую точность: 0.001
9.073053153782 0.000362135606 64.000000000000
```

Рисунок 27 – Проверка программы со второй точностью из файла

Заметим, что число итераций при изменении точности возрастает логарифмически, где число итераций – количество прямоугольников, которые разбивают график в точках касания.

Вывод: В ходе домашнего задания я поработал с погрешностями при вычислениях, написал программы, по-разному работающие с погрешностями. Узнал типы погрешностей и почему они возникают, научился работать с ними. Выявил отличия между float и double. Научился вычислять значения функций без сторонних библиотек, которые раскладываются в ряды с помощью счётных циклов и аккуратной работы с погрешностями. Научился вычислять длину кривой линии с определённой точностью.