Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы» направление подготовки: 09.03.04 – «Программная инженерия»

**Лабораторная работа №2.**

**«Численные методы решения нелинейных уравнений»**

Выполнил студент гр. РИС-24-2б

Долганов Даниил Вячеславович

Проверил:

Доц. Каф. ИТАС

Ольга Андреевна Полякова

(оценка) (подпись)

(дата)

г. Пермь, 2024

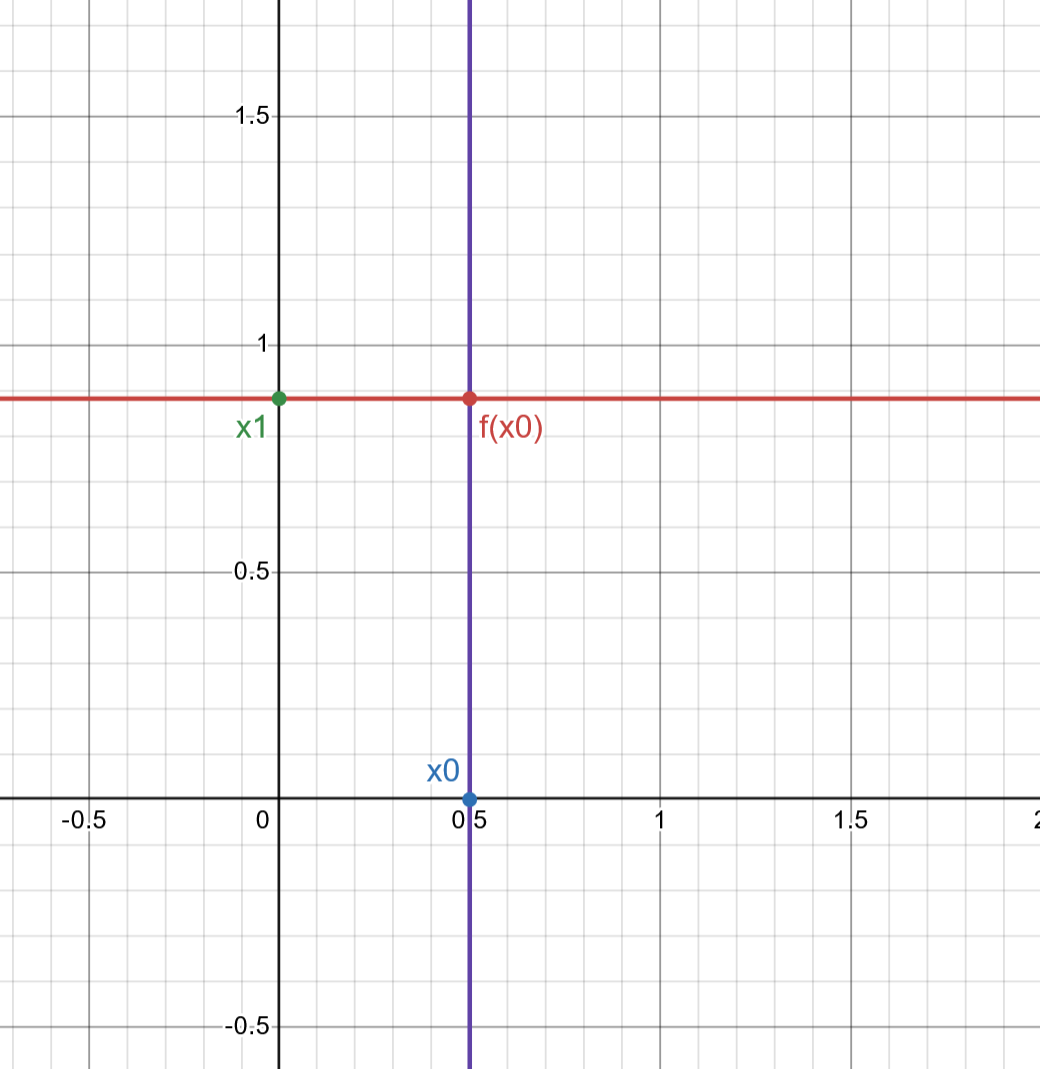
**Постановка задачи**

Дано уравнение f(x) = ex – ex – 2 = 0. Необходимо найти решение, использую методы: *итераций*, *половинного деления* и *Ньютона*. Отрезок содержащий корень [0; 1], точное значение корня 0,8814.

**Метод итераций**

1. *Геометрическая интерпретация задачи*

Для решения уравнения методом итераций необходимо преобразовать исходное уравнение из вида f(x) = 0 в x = f(µ). Преобразуя исходное уравнение получим, что x = ln(1 + ). Отсюда f(µ) = ln(1 + ). На отрезке [0; 1] выберем произвольное x0. Пусть x0 = 0,5. Тогда x1 = f(x0).

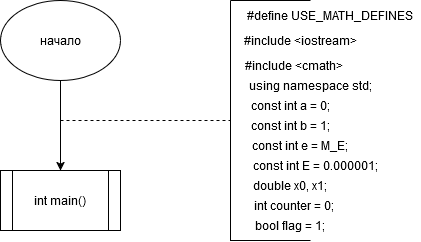


1 Нахождение по приближенному значению следующего приближения

Теперь найдем x2 = f(x1) и так до тех пор пока |xn - xn-1| > ɛ, где ɛ - необходимая нам точность решения. Пусть ɛ = 0.000001. Как только мы достигнем необходимой точности вычислений, решением будет xn или xn-1.

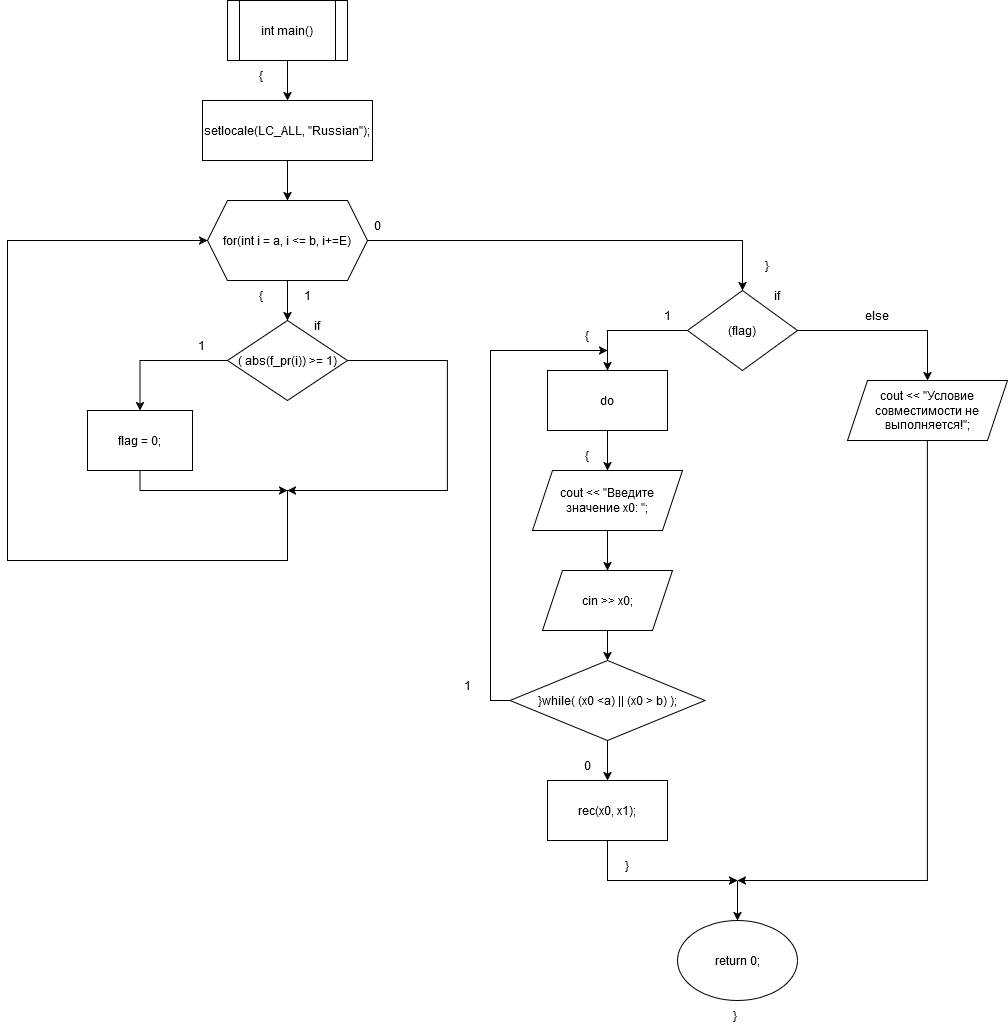
1. *Блок схема кода программы*

Для написания программы нам понадобится 4 функции: функция main, функция f(µ), производная от функции f(µ), а также рекурсивная функция, в которой будут вычисляться приближения x0 и x1. Пусть [a, b] – отрезок, на котором есть решение уравнения. a и b – константные значения. Также в качестве констант будет использовать ɛ, значение e. Зарезервируем память для x0 и x1. Заведем счетчик counter для будущего подсчета количества итераций и булевою переменную flag для будущей проверки на сходимость. Подключим директивы #define \_USE\_MATH\_DEFINES, #include <iostream>, #include <cmath>.



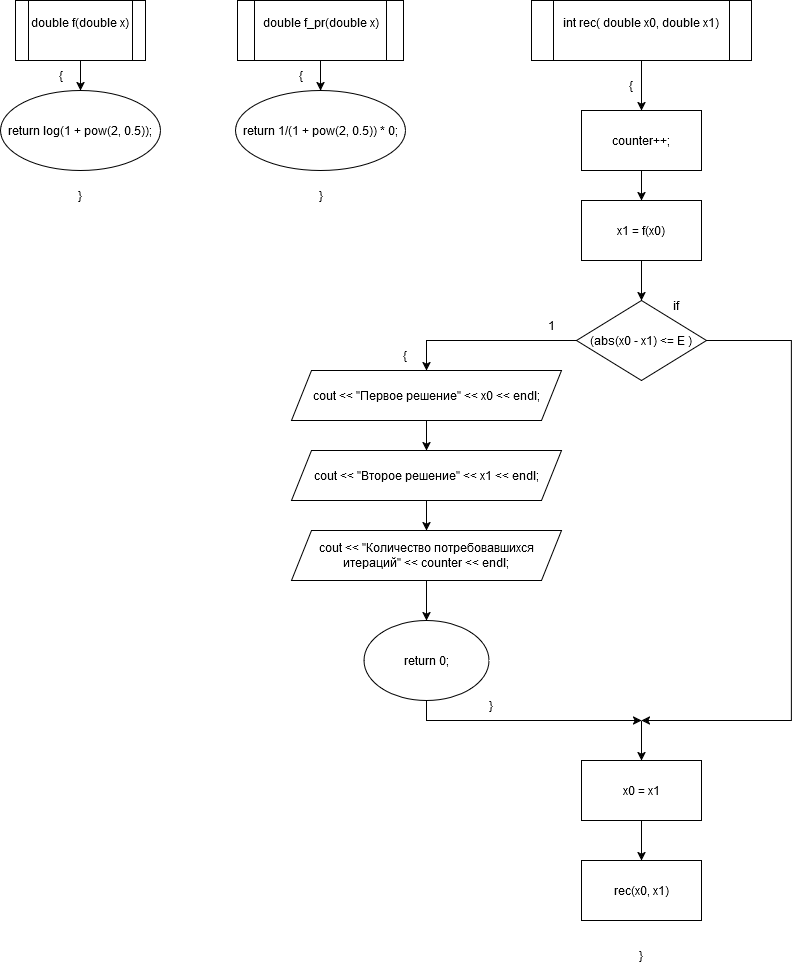
2 Начало алгоритма

Функция main будет проверять функцию f(µ) на сходимость, используя производную. Принимать от пользователя значение x0 и проверять его на корректность, а также функция main будет запускать рекурсивную функцию rec.



3 Функция int main()

Функция double f(double x) 🡺 f(µ). Функция double f\_pr(double x) 🡺 производная от f(µ). Функция int rec(double x0, double x1) – это рекурсивная функция, которая находит последующее приближение из изначального до тех пор, пока модуль разницы x0 и x1 больше чем ɛ, как только это условие нарушено выводит x0, x1 и количество потребовавшихся итераций.



4 Функции double f(double x), double f\_pr(double x) и double rec(double x0, double x1)

1. *Код программы*

#define \_USE\_MATH\_DEFINES

#include <iostream>

#include <cmath>

using namespace std;

double f(double x);

double f\_pr(double x);

int rec(double x0, double x1);

const int a = 0;

const int b = 1;

int counter = 0;

double x0, x1;

const double e = M\_E;

const double E = 0.000001;

bool flag = 1;

int main()

{

setlocale(LC\_ALL, "Russian");

for (double i = a; i <= b; i += E)

{

if (abs(f\_pr(i)) >= 1)

flag = 0;

}

if (flag)

{

do

{

cout << "Введите значение x0, лежащее на отрезке [" << a << ";" << b << "]: ";

cin >> x0;

} while ((x0 < a) || (x0 > b));

rec(x0, x1);

}

else { cout << "Условие совместимости функции невыполнено!"; }

return 0;

}

double f(double x)

{

return log(1 + pow(2, 0.5));

}

double f\_pr(double x)

{

return 1/(1 + pow(2, 0.5)) \* 0;

}

int rec(double x0, double x1)

{

counter++;

x1 = f(x0);

if (abs(x0 - x1) <= E)

{

cout << "Первое решение: " << x0 << endl;

cout << "Второе решение: " << x1 << endl;

cout << "Количество потребовавшийхся итераций: " << counter << endl;

return 0;

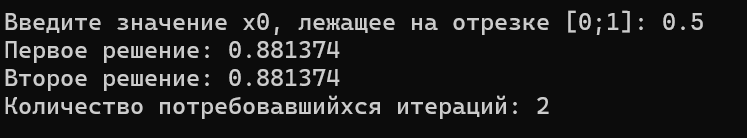
}

x0 = x1;

rec(x0, x1);

}

1. *Вывод программы*



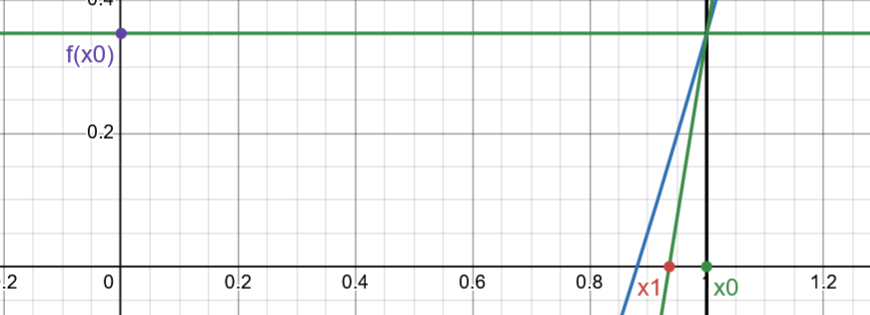
5 Вывод программы

Как можно увидеть из вывода программы ответ примерно равен точному, а значит программа сработала правильно.

**Метод Ньютона**

1. *Геометрическая интерпретация*

Для решения уравнения методом Ньютона необходимо представить исходное уравнение в виде f(x) = ex – e-x – 2. Далее необходимо определить точку x0 по правилу, если f(a)\*f ´´(a) > 0, то x0 = a, если f(b)\*f ´´(b) > 0, то x0 = b. a и b – концы отрезка [a;b], на котором находится решение уравнения. В нашем случае a = 0, b = 1, а x0 = b = 1. Далее из точки пересечения x0 и f(x0), из которой необходимо провести касательную y = k\*x + d, где k = f ´(x0). x1 -точка пересечения этой касательной с осью Ox.



6 Нахождение последующего значения x использую касательные

Повторять эту процедуру нужно до тех пор, пока |xn – xn-1| > ɛ. Где ɛ - заданная точность вычислений. Пусть ɛ = 0,000001. Как только мы достигнем необходимой точности вычислений, решением будет xn или xn-1.

Для нахождения последующего значения xn можно вывести формулу:

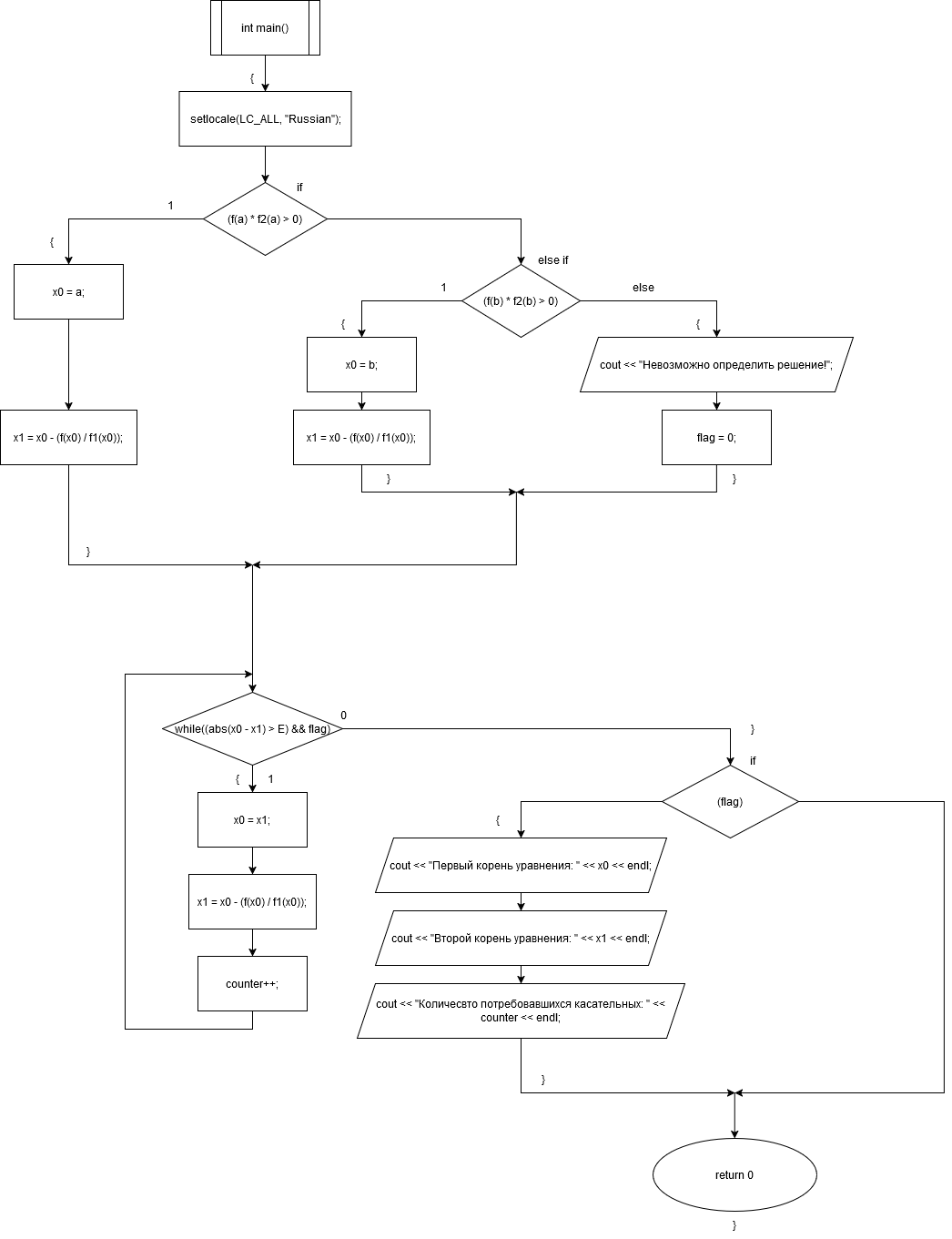
1. *Блок-схема кода программы*

Для написания программы нам понадобится 4 функции: функция main, в которой будут проходить основные вычисления, функция f(x), производная от функции f(x) и вторая производна от функции f(x). Пусть [a, b] – отрезок, на котором есть решение уравнения. a и b – константные значения. Также в качестве констант будет использоваться ɛ. Зарезервируем память для x0 и x1. Заведем счетчик counter для будущего подсчета количества касательных и булевою переменную flag. Подключим директивы #include <iostream>, #include <cmath>.

**

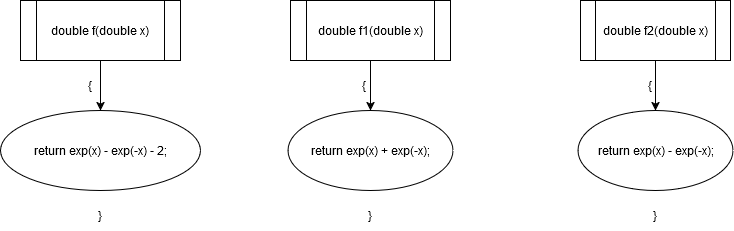
7 Начало алгоритма

Функция main будет определять точки x0 и x1 до тех пор пока |x0 – x1| > ɛ и выводить ответ.



8 Функция int main()

Функция double f(double x) 🡺 f(x). Функция double f1(double x) 🡺 производная от f(x). Функция double f2(double x) 🡺 вторая производная от f(x).



9 Функции double f(double x), double f1(double x) и double f2(double x)

1. *Код программы*

#include <iostream>

#include <cmath>

using namespace std;

double x0, x1;

const int a = 0;

const int b = 1;

const double E = 0.000001;

int counter = 1;

bool flag = 1;

double f(double x)

{

return exp(x) - exp(-x) - 2;

}

double f1(double x)

{

return exp(x) + exp(-x);

}

double f2(double x)

{

return exp(x) - exp(-x);

}

int main()

{

setlocale(LC\_ALL, "Russian");

if (f(a) \* f2(a) > 0)

{

x0 = a;

x1 = x0 - (f(x0) / f1(x0));

}

else if (f(b) \* f2(b) > 0)

{

x0 = b;

x1 = x0 - (f(x0) / f1(x0));

}

else

{

cout << "На заданном интервале невозможно определить корень уравнения!";

flag = 0;

}

while ((abs(x0 - x1) > E) && flag)

{

x0 = x1;

x1 = x0 - (f(x0) / f1(x0));

counter++;

}

if (flag)

{

cout << "Первый корень уравнения: " << x0 << endl;

cout << "Второй корень уравнения: " << x1 << endl;

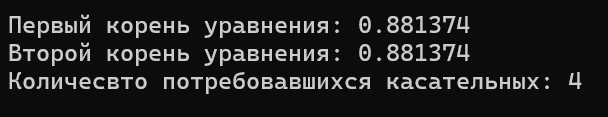
cout << "Количесвто потребовавшихся касательных: " << counter << endl;

}

return 0;

}

1. *Вывод программы*

**

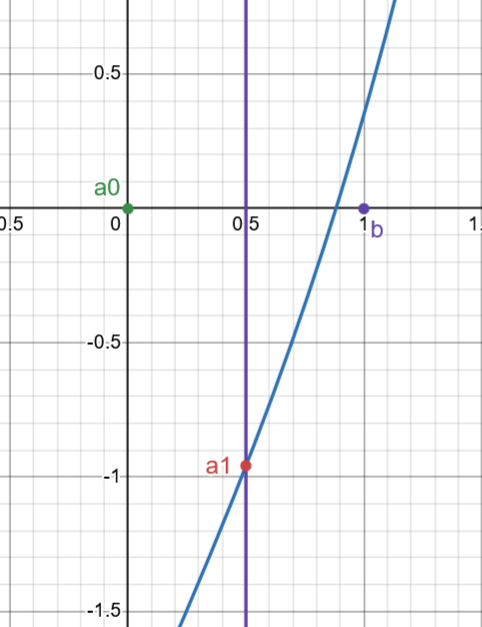
10 Вывод программы

Как можно увидеть из вывода программы ответ примерно равен точному, а значит программа сработала правильно.

**Метод половинного деления**

1. *Геометрическая интерпретация*

Для реализации данного метода нам понадобиться представить исходное уравнение в виде функции f(x) = ex – e-x – 2. Далее необходимо поделить исходный отрезок [a;b] пополам(), и отбросить часть отрезка в которой отсутствует корень. Если f(a)\*f(с) > 0, то a = c, иначе b = c. В нашем случае a = 0, b = 1, отсюда c = 0,5.

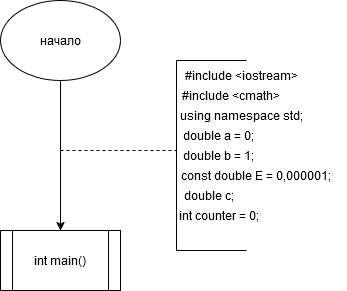


11 отбрасывание отрезков, на которых отсутствует корень

Повторять эту процедуру нужно до тех пор, пока |a – b| > ɛ. Где ɛ - заданная точность вычислений. Пусть ɛ = 0,000001. Как только мы достигнем необходимой точности вычислений, решением будет a или b.

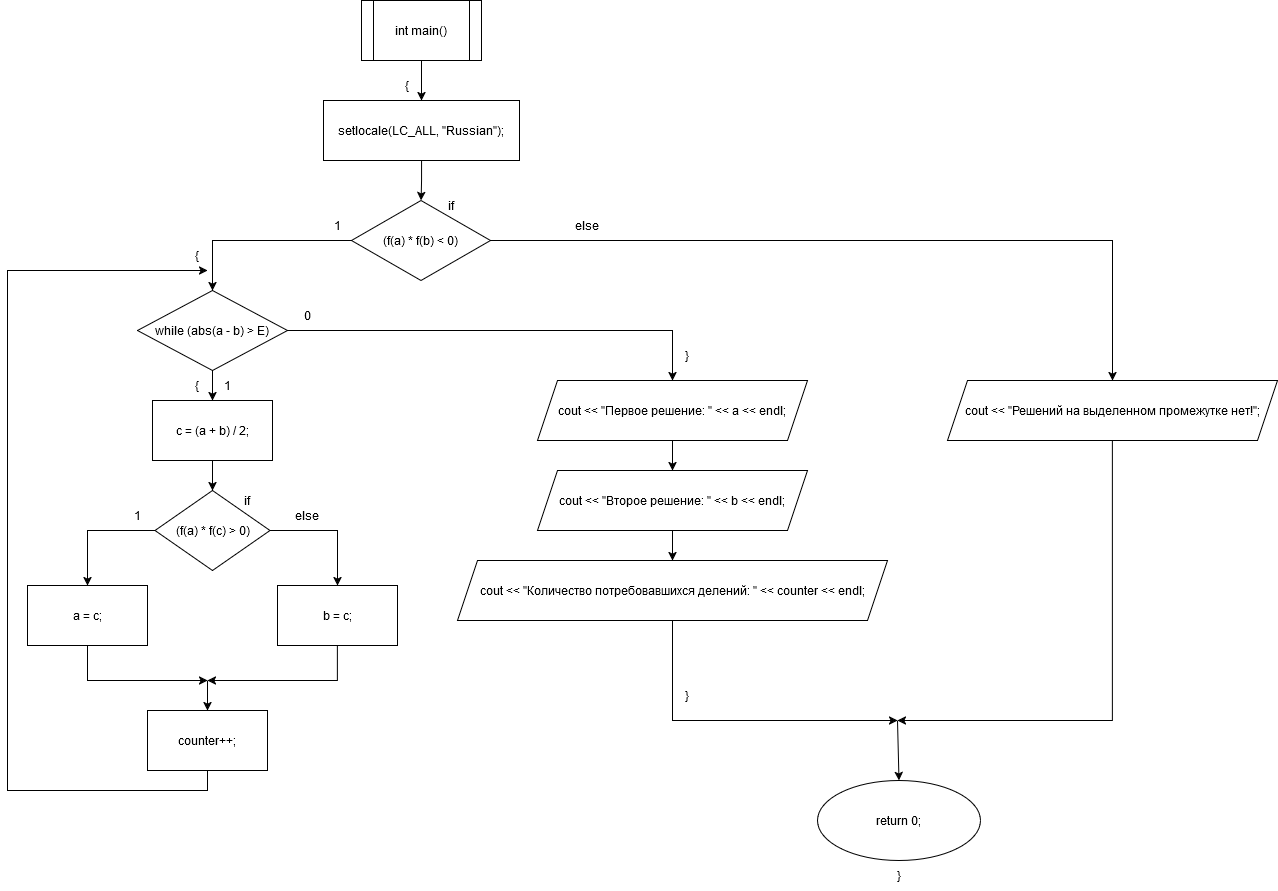
1. *Блок-схема кода программы*

Для написания программы нам понадобится 2 функции: функция main, в которой будут проходить основные вычисления и функция f(x). Пусть [a, b] – отрезок, на котором есть решение уравнения. Заведем две переменные a = 0 и b = 1. В качестве константы будет использоваться ɛ. Зарезервируем память для переменной c. Заведем счетчик counter для будущего подсчета количества потребовавшихся делений. Подключим директивы #include <iostream>, #include <cmath>.

**

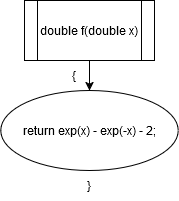
12 Начало алгоритма

Функция main будет находить новые значения a и b, до тех пор пока |a – b| > ɛ.



13 Функция main

Функция double f(double x) 🡺 f(x).



14 Функция double f(double x)

1. *Код программы*

#include <iostream>

#include <cmath>

using namespace std;

double f(double x)

{

return exp(x) - exp(-x) - 2;

}

double a = 0;

double b = 1;

const double E = 0.000001;

double c;

int counter = 0;

int main()

{

setlocale(LC\_ALL, "Russian");

if (f(a) \* f(b) < 0)

{

while (abs(a - b) > E)

{

c = (a + b) / 2;

if (f(a) \* f(c) > 0)

a = c;

else b = c;

counter++;

}

cout << "Первое решение: " << a << endl;

cout << "Второе решение: " << b << endl;

cout << "Количество потребовавшихся делений: " << counter << endl;

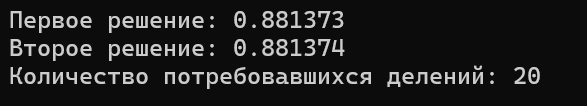
}

else cout << "Решений на выделенном промежутке нет!";

return 0;

}

1. *Вывод программы*



15 Вывод программы

Как можно увидеть из вывода программы ответ примерно равен точному, а значит программа сработала правильно.

**Вывод по проделанной работе**

Как видно по ходу работы, все программы работают верно, а значит все задачи решены.

Ссылка на гитхаб: