ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»  
(РУТ (МИИТ))

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

ОТЧЁТ  
О ЛАБОРАТОРНОЙ РАБОТЕ №3

По дисциплине «Языки программирования»

ВАРИАНТ 8

Выполнил: ст. гр. ТКИ - 142

Лазарев Андрей Александрович

Проверил: к.т.н., доц. Васильева М. А.

(Проверил: к.т.н, доц. Балакина Е. П.)

Москва 2023

Оглавление

[1. Задание 3.1 3](#_Toc153558724)

[1.1. Формулировка задания 3](#_Toc153558725)

[1.2. Блок-схема алгоритма 4](#_Toc153558726)

[1.3. Код задания 3.1 7](#_Toc153558727)

[1.4. Решение тестовых примеров 9](#_Toc153558728)

[1.5. Расчёт тестового примера в Excel 10](#_Toc153558729)

[1.6. Зачёт задания в GitHub 10](#_Toc153558730)

[2. Задание 3.2 11](#_Toc153558731)

[2.1. Формулировка задания 11](#_Toc153558732)

[2.2. Блок-схема алгоритма 11](#_Toc153558733)

[2.3. Код задания 3.2 11](#_Toc153558734)

[2.4. Решение тестовых примеров 11](#_Toc153558735)

[2.5. Решение тестовых примеров в Excel 11](#_Toc153558736)

[2.6. Зачёт задания в GitHub 11](#_Toc153558737)

[3. Задание 3.3 11](#_Toc153558738)

[3.1. Формулировка задания 11](#_Toc153558739)

[3.2. Блок-схема алгоритма 12](#_Toc153558740)

[3.3. Код задания 3.3 16](#_Toc153558741)

[3.4. Решение тестового примера 19](#_Toc153558742)

[3.5. Решение тестовых примеров в Excel 20](#_Toc153558743)

[3.6. Зачёт задания в GitHub 21](#_Toc153558744)

# Задание 3.1

* 1. Формулировка задания

Протабулировать заданную в таблице функцию. Использовать данные в таблице значения шага и интервала в качестве ввода пользователя для решения тестового примера. При невозможности расчёта функции в конкретной точке выводить её значение и надпись, означающую отсутствие решения.

Таблица 1 – Формулировка задания 3.1

|  |  |  |
| --- | --- | --- |
| Вариант | Функция | Константы |
| 8 |  |  |

* 1. Блок-схема алгоритма

Блок-схема алгоритма представлена на рисунках (Рисунок 1, Рисунок 2, Рисунок 3, Рисунок 4, Рисунок 5)

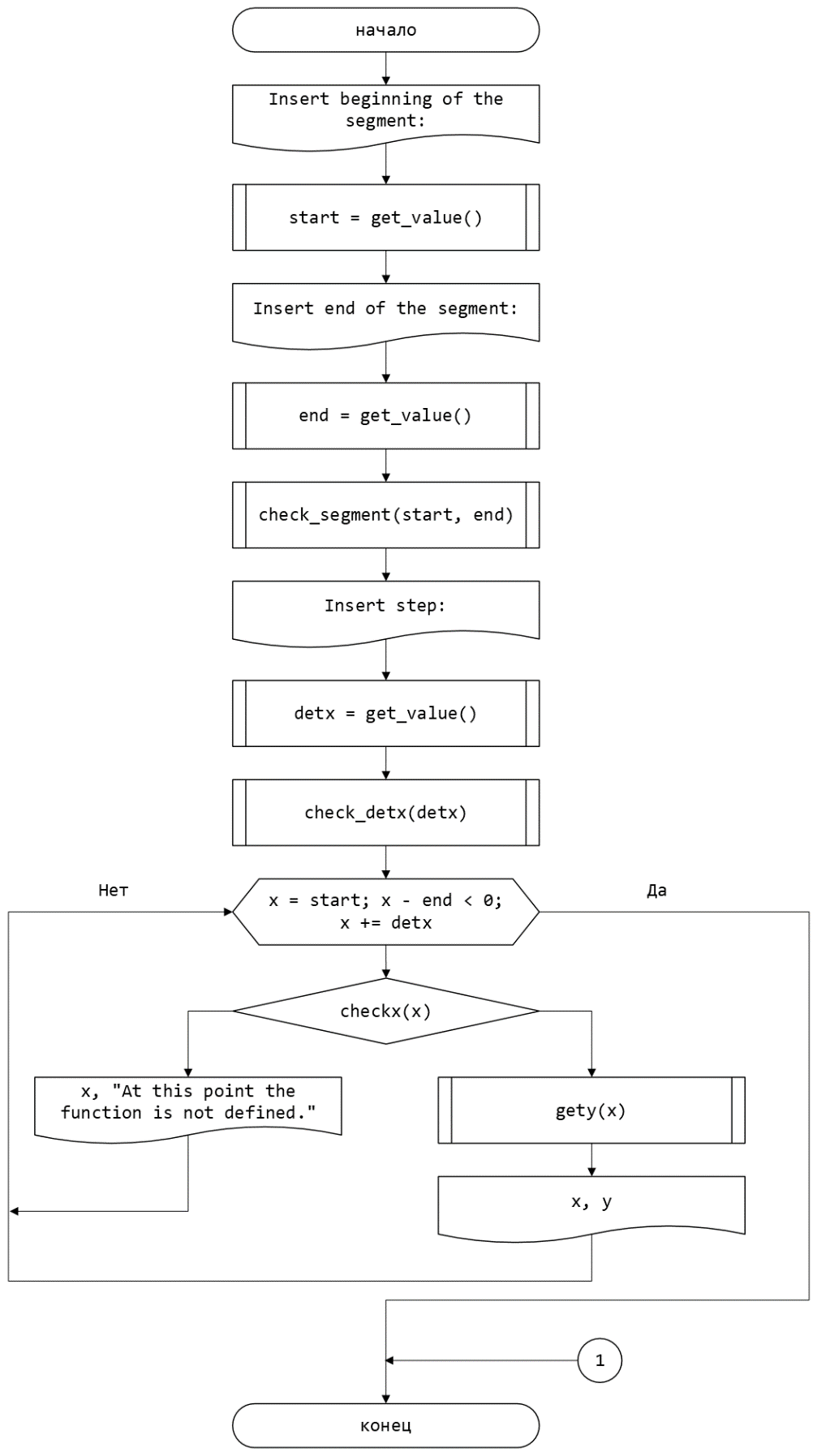


Рисунок 1 – Блок-схема функции main()

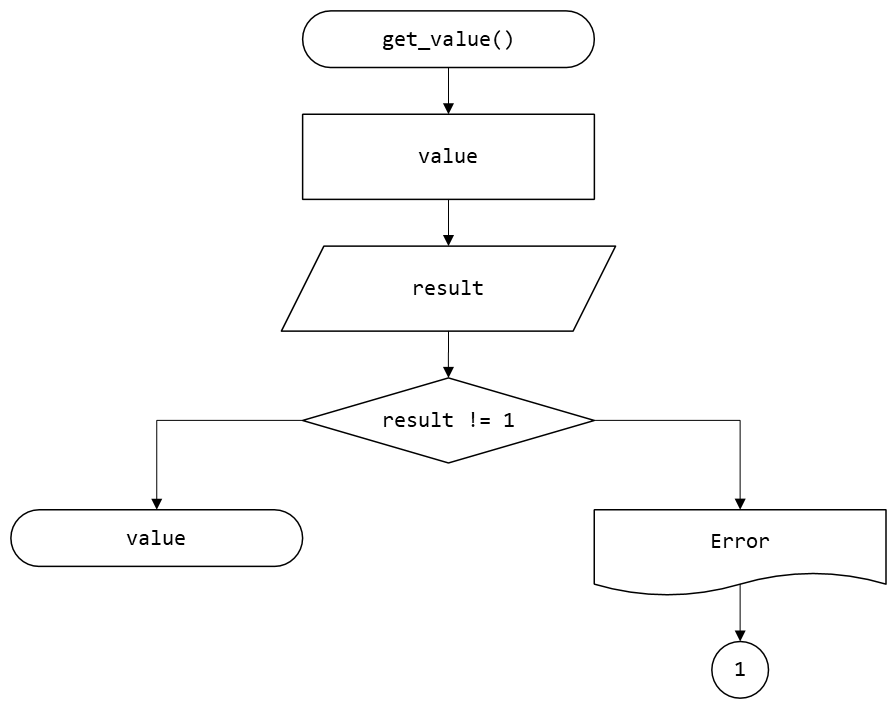


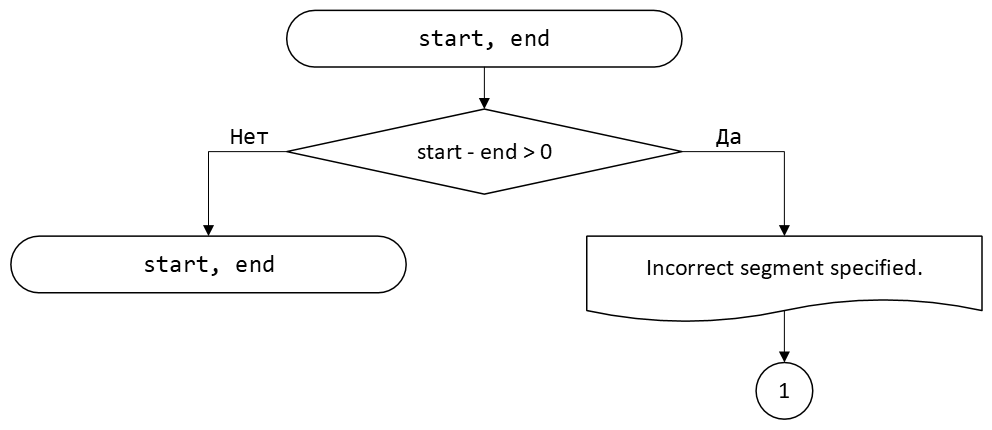
Рисунок 2 – Блок-схема функции get\_value()

Рисунок 3 – Блок-схема к функции check\_segment(start, end)

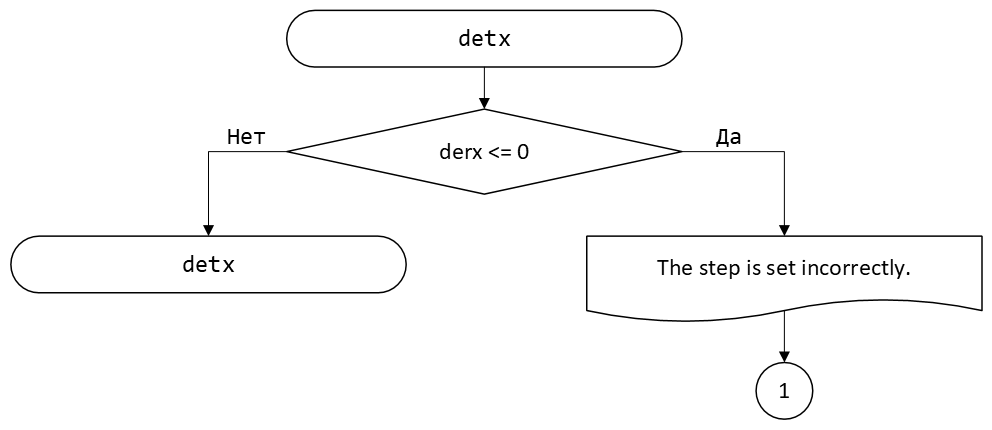


Рисунок 4 – Блок-схема функции check\_detx(detx)

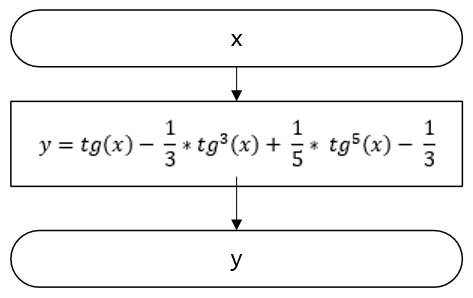


Рисунок 5 – Блок-схема функции getY(x)

* 1. Код задания 3.1

#include <stdio.h>

#include <math.h>

#include <stdbool.h>

#include <float.h>

#include <stdlib.h>

#include <errno.h>

/\*\*

\* @brief Функция расчета по заданной формуле.

\* @param x Аргумент функции.

\* @return Возвращает значение фенкции.

\*/

double gety(double x);

/\*\*

\* @brief Функция проверки переменной x.

\* @param x Аргумент функции.

\* @return Возвращает истину в случае успеха.

\*/

bool checkx(double x);

/\*\*

\* @brief Функция проверки отрезка на правильность.

\* @param start Аргумент функции.

\* @param end Аргумент функции.

\* @return Возвращает ошибку в случае успеха.

\*/

double check\_segment(double start, double end);

/\*\*

\* @brief Функция проверки шага на правильность.

\* @param detx Аргумент функции.

\* @return Возвращает ошибку в случае успеха.

\*/

double check\_detx(double detx);

/\*\*

\* @brief Функция проверки ввода на правильность.

\* @return Возвращает значение в случае успеха.

\*/

double get\_value();

/\*\*

\* @brief Точка входа в программу.

\* @return Возвращает 0 в случае успеха.

\*/

int main()

{

printf("%s", "Insert beginning of the segment: ");

double start = get\_value();

printf("%s", "Insert end of the segment: ");

double end = get\_value();

check\_segment(start, end);

printf("%s", "Insert step: ");

double detx = get\_value();

check\_detx(detx);

for (double x = start; x - end < DBL\_EPSILON; x += detx)

{

if (checkx(x))

{

printf("x = %lf y = %lf\n", x, gety(x));

}

else

{

printf("x = %lf At this point the function is not defined.", x);

}

}

return 0;

}

bool checkx(double x)

{

if (fabs(cos(x)) > -DBL\_EPSILON)

{

return true;

}

return false;

}

double gety(double x)

{

return tan(x) - (1.0 / 3) \* pow(tan(x), 3) + (1.0 / 5) \* pow(tan(x), 5) - (1.0 / 3);

}

double check\_segment(double start, double end)

{

if (start - end > -DBL\_EPSILON)

{

printf("%s", "Incorrect segment specified.");

abort();

}

}

double check\_detx(double detx)

{

if (detx <= DBL\_EPSILON)

{

printf("%s", "The step is set incorrectly.");

abort();

}

}

double get\_value()

{

double value;

int result = scanf\_s("%lf", &value);

if (result != 1)

{

errno = EIO;

perror("Error");

abort();

}

return value;

}

* 1. Решение тестовых примеров

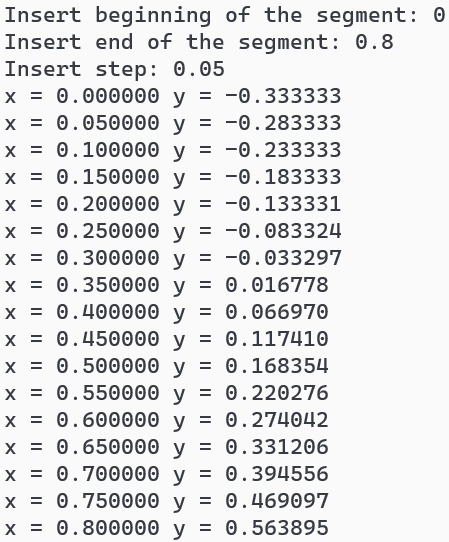


Рисунок 6 – Решение тестового примера

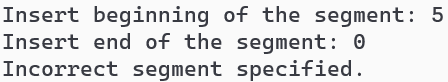


Рисунок 7 – Вывод программы, когда начало сегмента больше конца



Рисунок 8 – Вывод программы, когда начало сегмента – буква

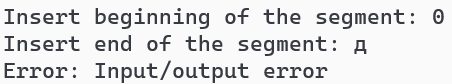


Рисунок 9 – Вывод программы, когда конец отрезка – буква

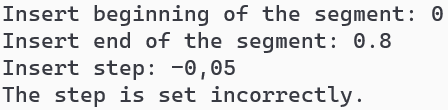


Рисунок 10 – Вывод программы, когда шаг функции отрицательный

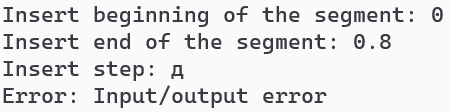


Рисунок 11 – Вывод программы, когда шаг функции – буква

* 1. Расчёт тестового примера в Excel

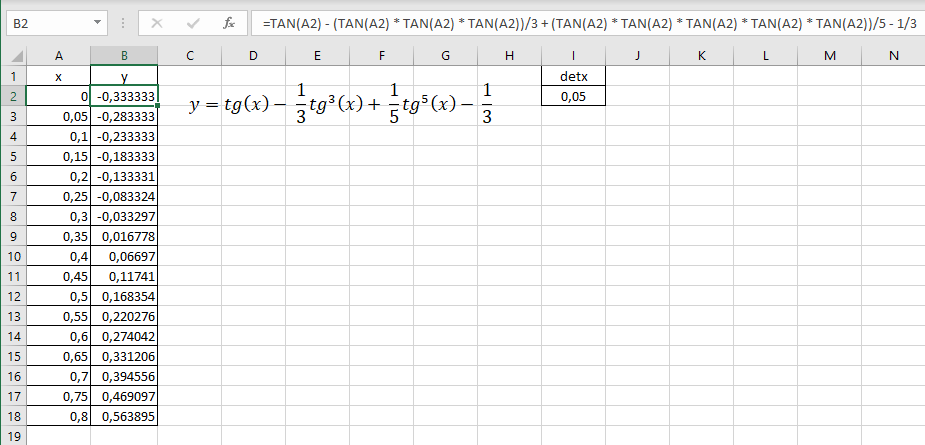


Рисунок 12 – Решение тестового примера в Excel

* 1. Зачёт задания в GitHub

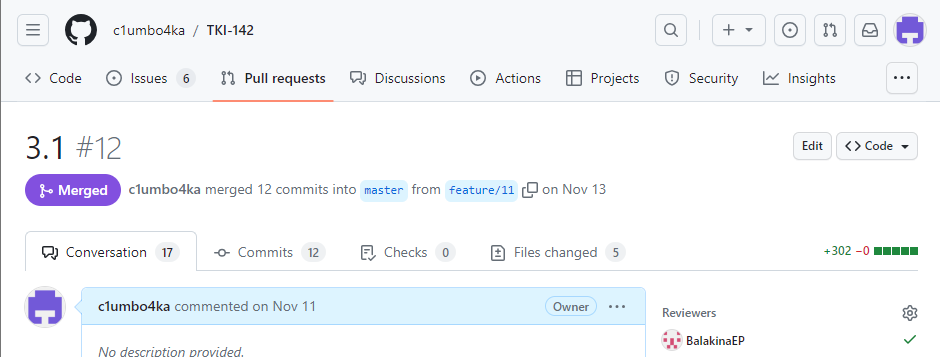


Рисунок 13 – Зачёт задания в GitHub

# Задание 3.2

* 1. Формулировка задания

1. вычислить сумму первых *n* членов последовательности (*k* = 1, 2, 3 ..., *n*).
2. вычислить сумму всех членов последовательности, не меньших заданного числа *e*.

Таблица 2 – Формулировка задания 3.2

|  |  |
| --- | --- |
| **Вар** | **Ряд** |
| 8 |  |

* 1. Блок-схема алгоритма

Блок-схема алгоритма представлена на рисунках (Рисунок 14, Рисунок 15, Рисунок 16, Рисунок 17, Рисунок 18, Рисунок 19)

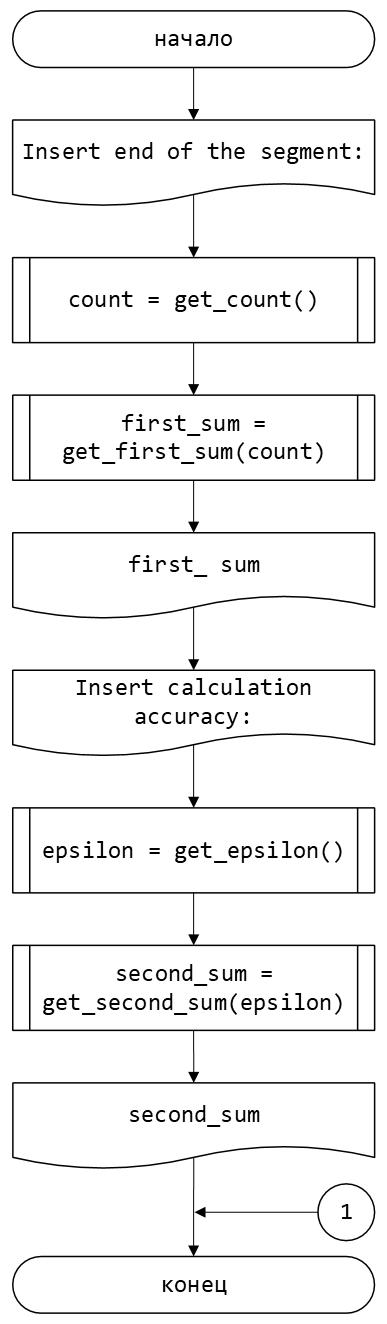


Рисунок 14 – Блок-схема функции main()

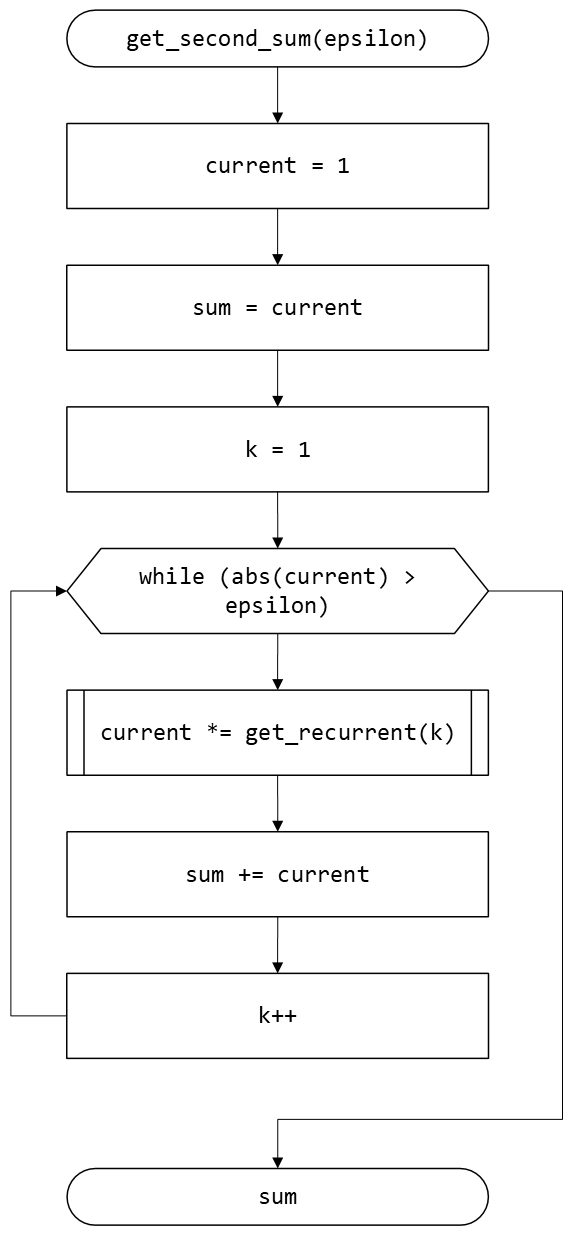


Рисунок 15 – Блок-схема функции get\_second\_sum(epsilon)

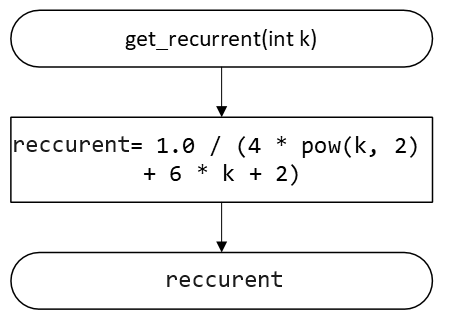


Рисунок 16 – Блок-схема функции get\_reccurent(k)

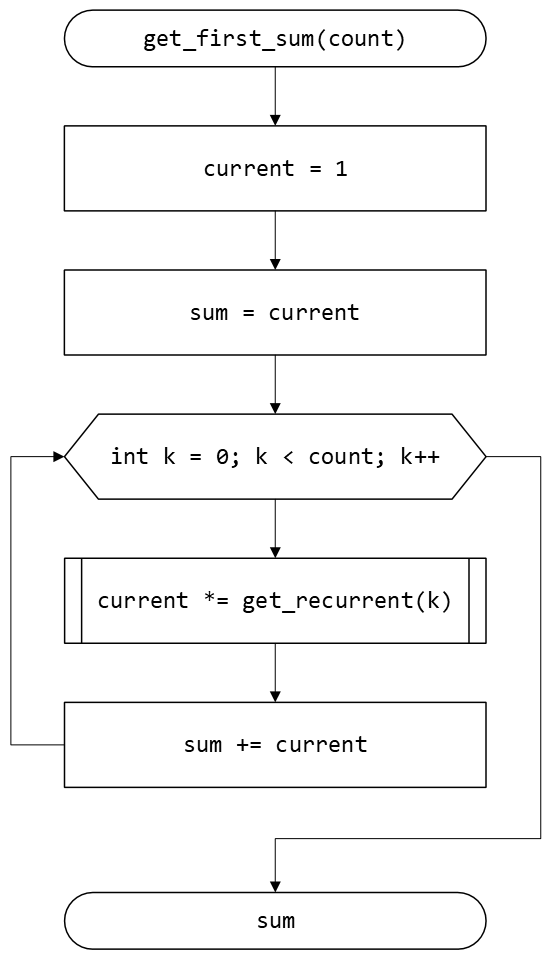


Рисунок 17 – Блок-схема функции get\_first\_sum(count)

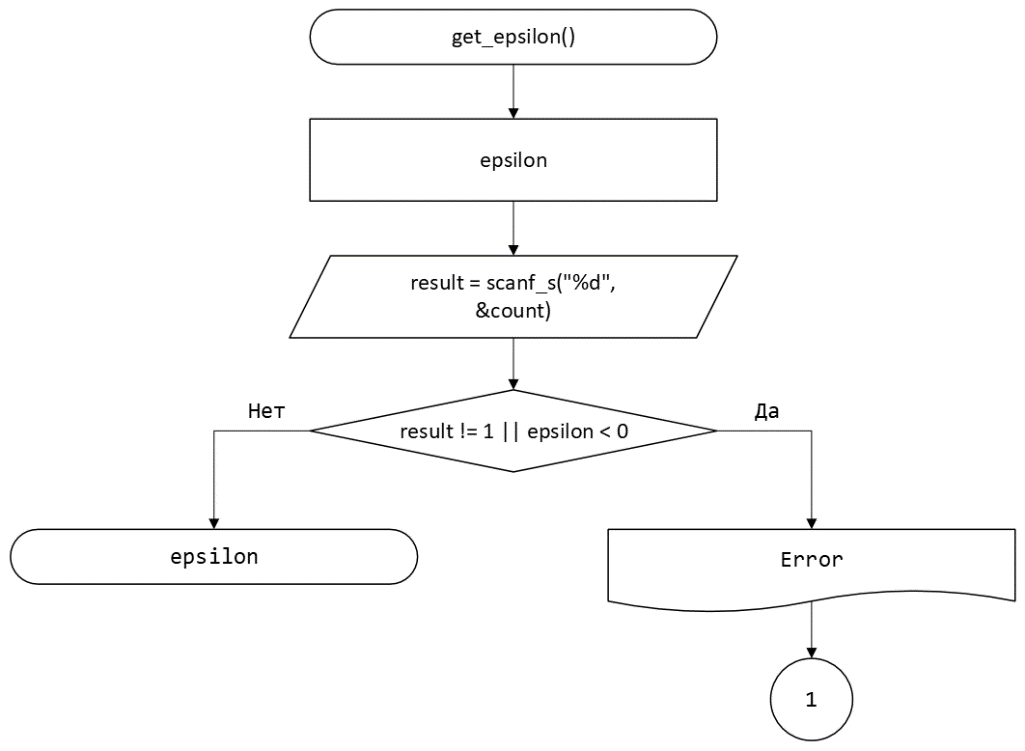


Рисунок 18 – Блок-схема функции get\_epsilon()

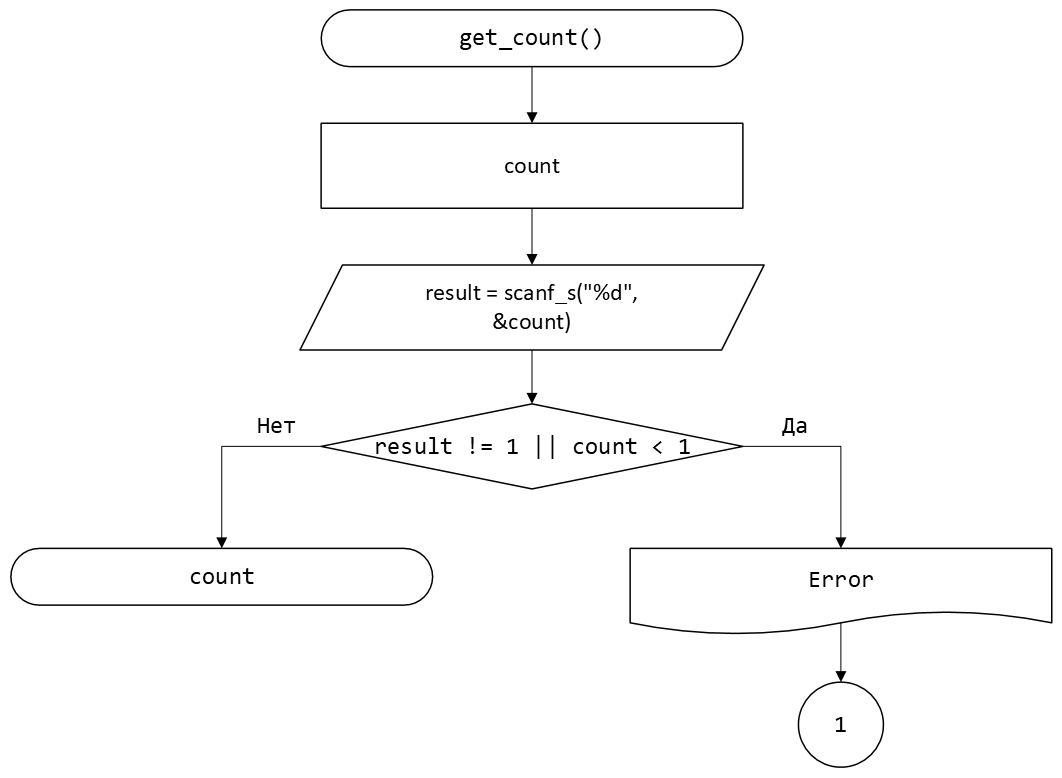


Рисунок 19 – Блок-схема функции get\_count()

* 1. Код задания 3.2

#include <stdio.h>

#include <math.h>

#include <float.h>

#include <errno.h>

#include <stdlib.h>

/\*\*

\* @brief Функция считывающая количество членов последовательности.

\* @remarks Экстренное завершение программы, в случае неправильного ввода.

\* @return Количество членов последовательности.

\*/

int get\_count();

/\*\*

\* @brief Функция расчитывающая сумму членов последовательности.

\* @param count - количество членов последовательности.

\* @return Сумма последовательности.

\*/

double get\_first\_sum(int count);

/\*\*

\* @brief Функция расчитывающая рекурентный член последовательности.

\* @param k - номер члена последовательности.

\* @return Значение рекурентного члена последовательности.

\*/

double get\_recurrent(int k);

/\*\*

\* @brief Функция ввода и проверки на правильность точности вычислений.

\* @return Точность вычислений.

\*/

double get\_epsilon();

/\*\*

\* @brief Функция вычисления суммы членов последовательности, не меньших epsilon.

\* @param epsilon - точность вычислений.

\* @return Сумма.

\*/

double get\_second\_sum(const double epsilon);

/\*\*

\* @brief Точка входа в программу.

\* @return Возвращает 0 в случае успеха.

\*/

int main()

{

printf("%s", "Insert end of the segment: ");

int count = get\_count();

printf\_s("Sum of %d sequence terms: %lf \n", count + 1, get\_first\_sum(count));

printf("Insert calculation accuracy: ");

const double epsilon = get\_epsilon();

printf("The sum of the terms of the sequence not less than calculation accuracy: %lf", get\_second\_sum(epsilon));

return 0;

}

int get\_count()

{

int count;

int result = scanf\_s("%d", &count);

if (result != 1 || count < 1)

{

errno = EIO;

perror("Error");

abort();

}

return count;

}

double get\_first\_sum(int count)

{

double current = 1;

double sum = current;

for (int k = 0; k < count; k++)

{

current \*= get\_recurrent(k);

sum += current;

}

return sum;

}

double get\_recurrent(int k)

{

return 1.0 / (4 \* pow(k, 2) + 6 \* k + 2);

}

double get\_epsilon()

{

double epsilon;

double result = scanf\_s("%lf", &epsilon);

if (result != 1 || epsilon < DBL\_EPSILON)

{

errno = EIO;

perror("Error");

abort();

}

return epsilon;

}

double get\_second\_sum(const double epsilon)

{

double current = 1;

double sum = current;

int k = 1;

while (current >= epsilon)

{

current \*= get\_recurrent(k);

sum += current;

k++;

}

return sum;

}

* 1. Решение тестовых примеров

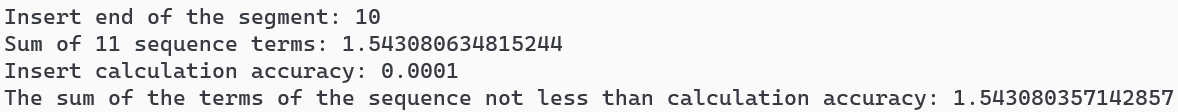


Рисунок 20 – Решение тестового примера

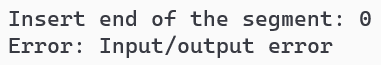


Рисунок 21 – Вывод программы, когда количество членов последовательности неположительное



Рисунок 22 – Вывод программы, когда количество членов последовательности – буква

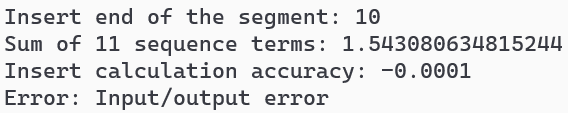


Рисунок 23 – Вывод программы, когда точность вычислений отрицательная

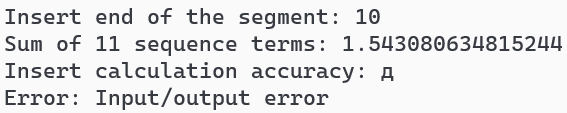


Рисунок 24 – Вывод программы, когда точность вычислений – буква

* 1. Решение тестовых примеров в Excel

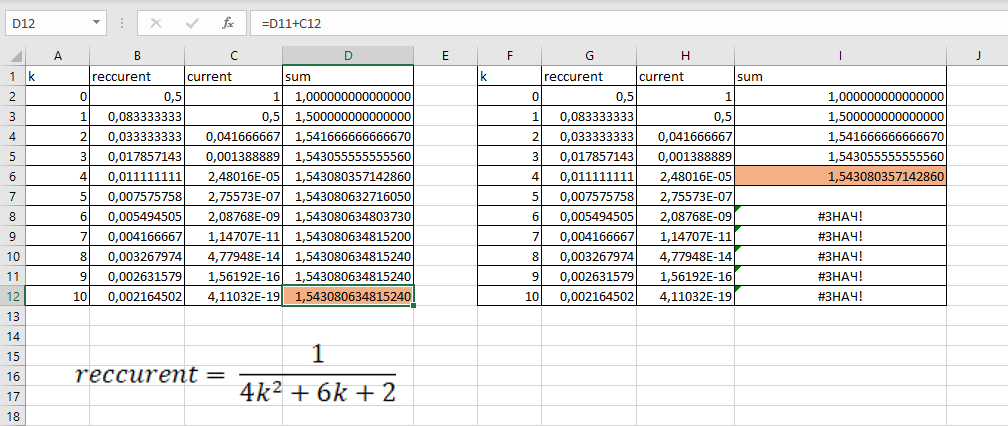


Рисунок 25 – Расчёт суммы первых 10 членов последовательности в Excel

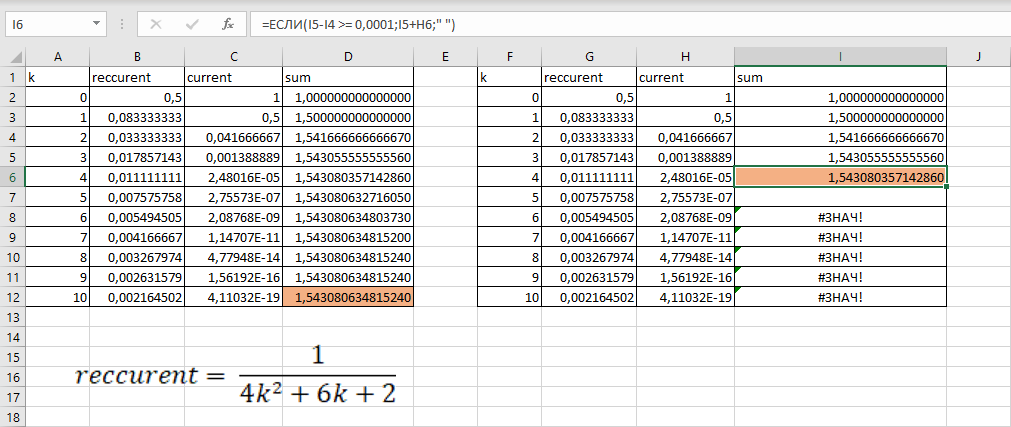


Рисунок 26 – Расчёт суммы членов последовательности, не меньших заданного числа e

* 1. Зачёт задания в GitHub

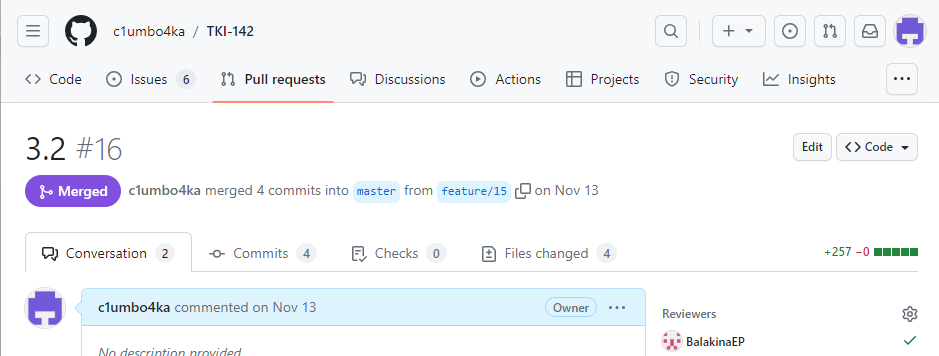


Рисунок 27 – Зачёт задания в GitHub

# Задание 3.3

* 1. Формулировка задания

Протабулировать заданную функцию и сумму функционального ряда разложения этой функции на интервале [*a,b*] и с шагом *h* (шаг и интервал задается в константах). Функциональнй ряд вычисляется по соответствующей рекуррентной формуле с заданной точностью *ɛ*. В результате показать три столбца: значение аргумента, значение функции в данной точке и значение суммы ряда, вычисленное с заданной точностью в данной точке. Два последних столбца должны иметь близкие результаты.

Таблица 3 – Формулировка задания 3.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вар | Функция y | Сумма | Интервал | ɛ |
| 8 |  |  |  | 10-4 |

* 1. Блок-схема алгоритма

Блок-схема алгоритма представлена на рисунках (Рисунок 28, Рисунок 29, Рисунок 30, Рисунок 31, Рисунок 32, Рисунок 33, Рисунок 34, Рисунок 35, Рисунок 36)

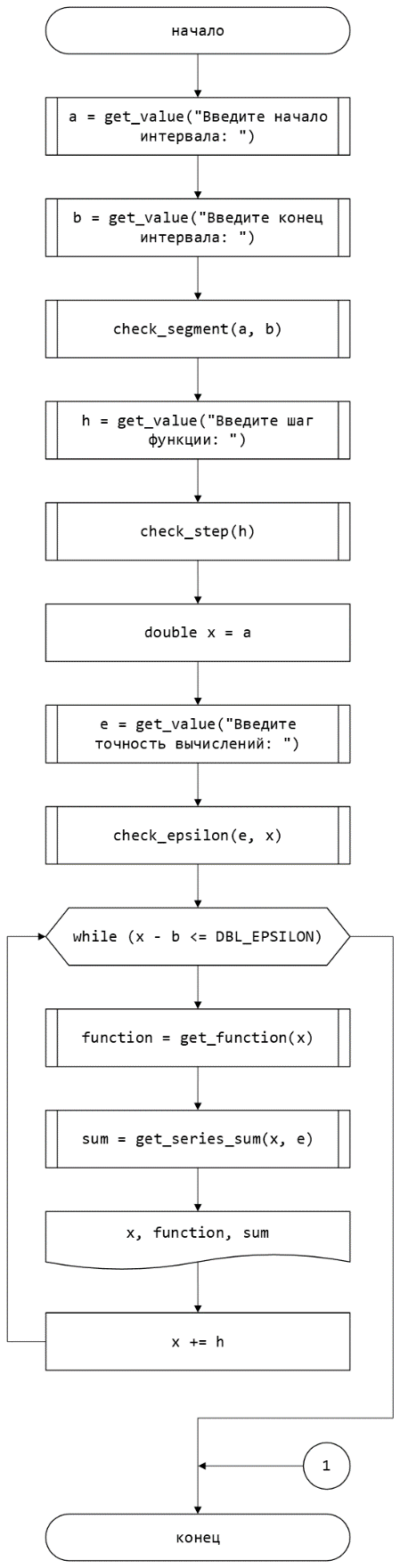


Рисунок 28 – Блок-схема функции main()

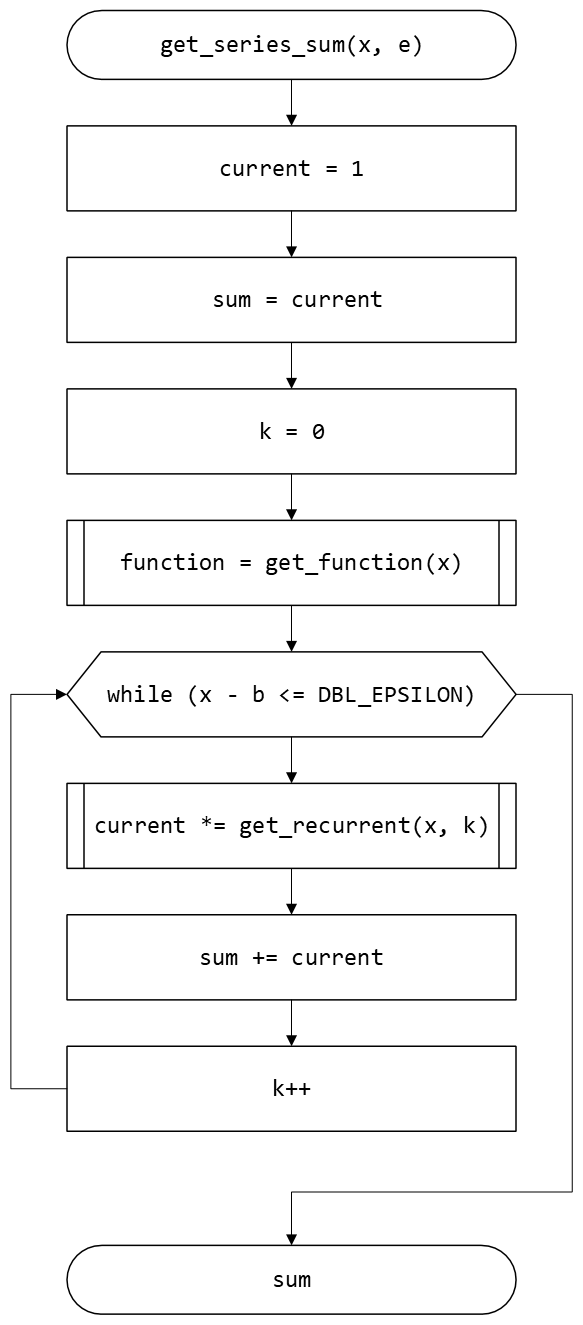


Рисунок 29 – Блок-схема функции get\_series\_sum(x, e)

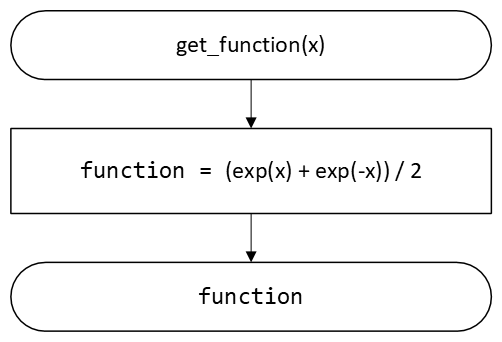


Рисунок 30 – Блок-схема функции get\_function(x)

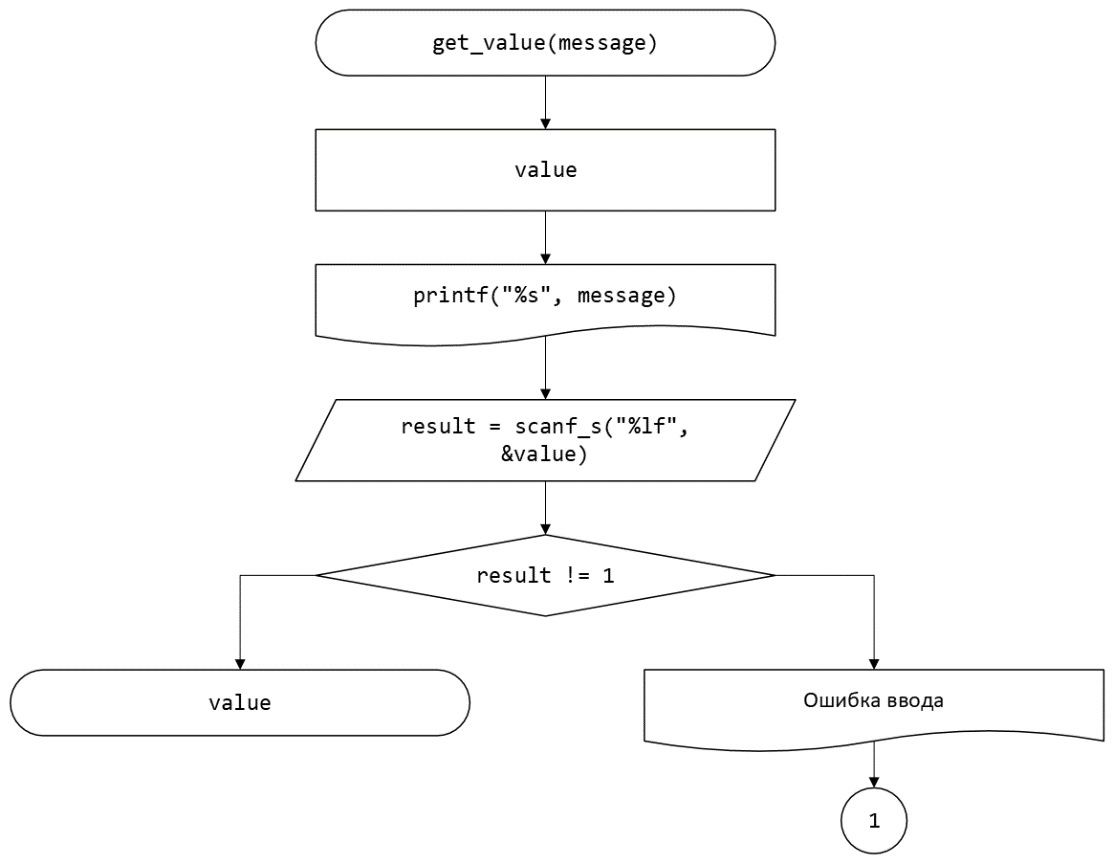


Рисунок 31 – Блок-схема функции get\_value(message)

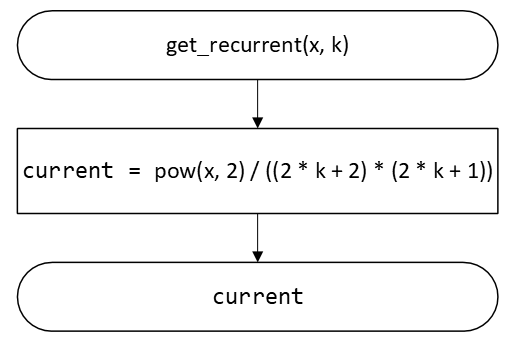


Рисунок 32 – Блок-схема функции get\_reccurent(x, k)

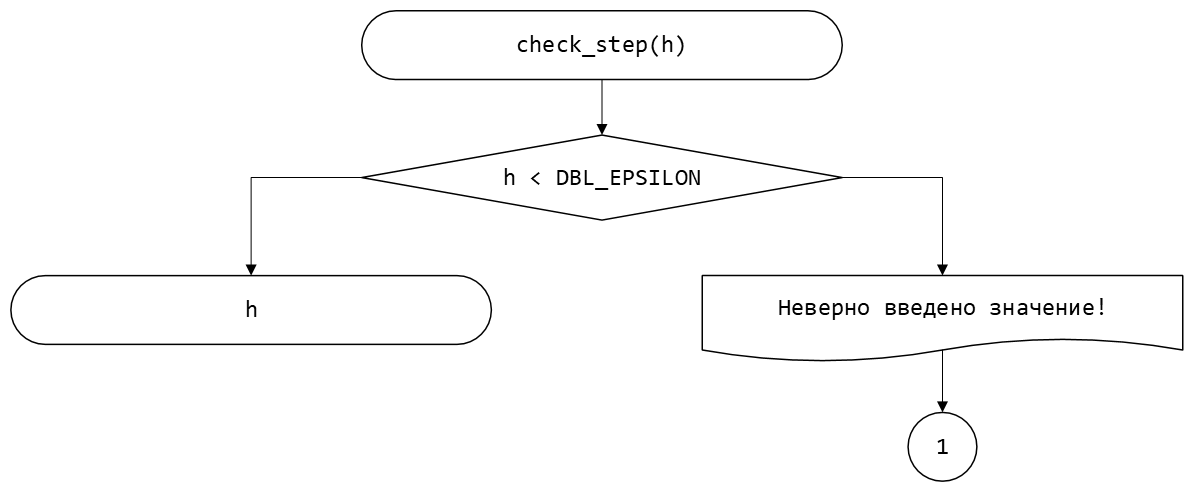


Рисунок 33 – Блок-схема функции check\_step(h)

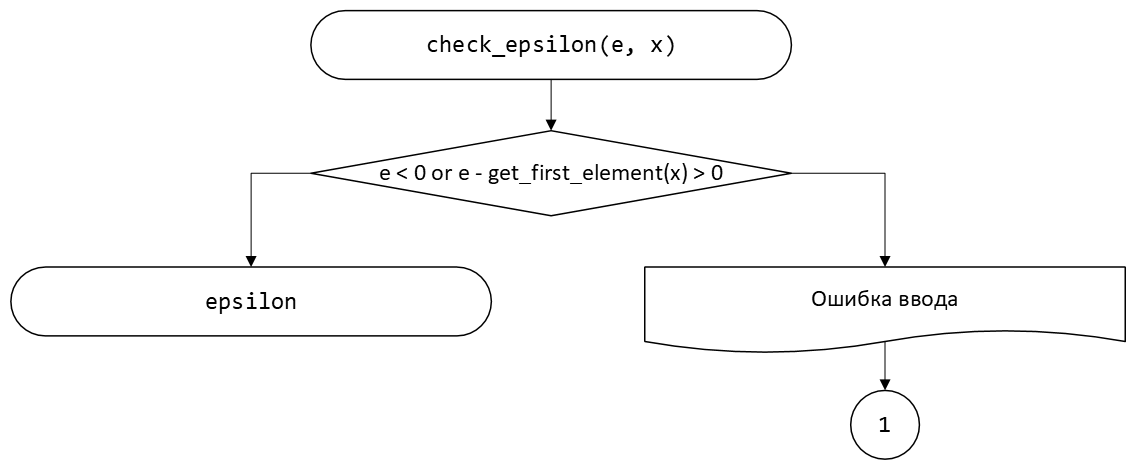


Рисунок 34 – Блок-схема функции check\_epsilon(e, x)

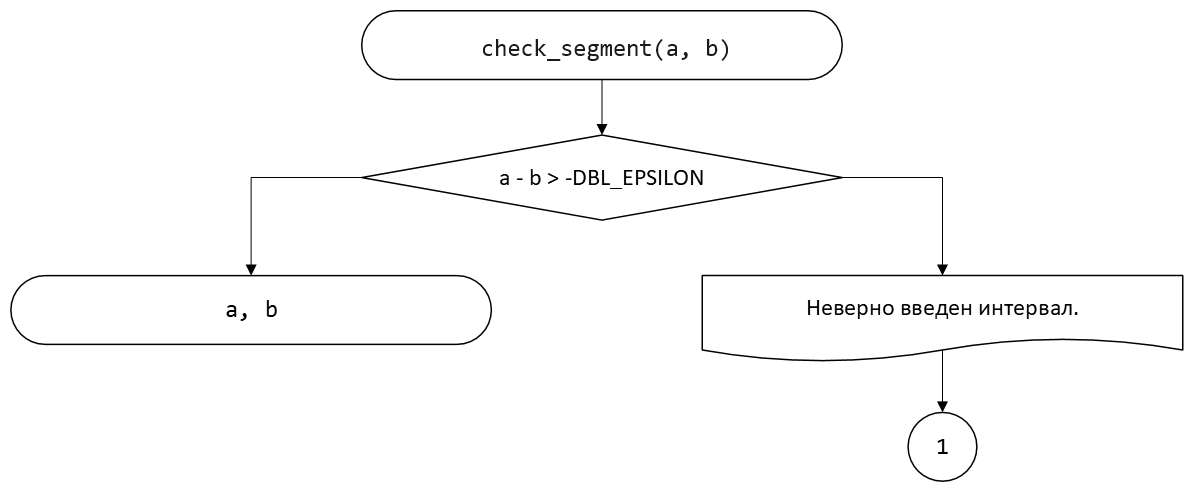


Рисунок 35 – Блок-схема функции check\_segment(a, b)

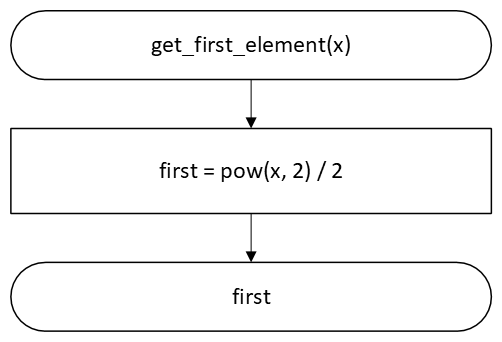


Рисунок 36 – Блок-схема функции get\_first\_element(x)

* 1. Код задания 3.3

#include <stdio.h>

#include <locale.h>

#include <errno.h>

#include <math.h>

#include <stdlib.h>

#include <float.h>

/\*\*

\* @brief Функция для вычисления рекуррентного члена ряда.

\* @param x Параметр функции, аппроксимируемой рядом.

\* @param k Индекс члена ряда.

\* @return Значение рекуррентного члена.

\*/

double get\_recurrent(double x, double k);

/\*\*

\* @brief Функция для вычисления суммы ряда.

\* @param x Показатель степени.

\* @return Значение 3 в степени x.

\*/

double get\_series\_sum(double x, double e);

/\*\*

\* @brief Функция для вычисления значения функции.

\* @param x - аргумент функции.

\* @return Значение функции.

\*/

double get\_function(double x);

/\*\*

\* @brief Функция проверки ввода шага на правильность.

\* @param h - шаг функции.

\* @remarks Экстренное завершение программы, в случае неправильного ввода.

\*/

void check\_step(double h);

/\*\*

\* @brief Функция проверки шага на правильность.

\* @param message - сообщение пользователю.

\* @remarks Экстренное завершение программы, в случае неправильного ввода.

\* @return Возвращает значение в случае успеха.

\*/

double get\_value(const char\* message);

/\*\*

\* @brief Функция проверки отрезка на существование.

\* @param a - начало отрезка.

\* @param b - конец

отрезка.

\*/

void check\_segment(const double a, const double b);

/\*\*

\* @brief Функция проверки на правильность точности вычислений.

\* @remarks Экстренное завершение программы, в случае неправильного ввода.

\* @return Возвращает значение в случае успеха.

\*/

void check\_epsilon(double e, double x);

/\*\*

\* @brief Функция расчёта первого элемента последовательности.

\* @param x - аргумент функции.

\* @return Возвращает значение первого элемента последовательности.

\*/

double get\_first\_element(double x);

/\*\*

\* @brief Точка входа в программу.

\* @return Возвращает 0 в случае успеха.

\*/

int main()

{

setlocale(LC\_ALL, "RU");

const double a = get\_value("Введите начало интервала: ");

const double b = get\_value("Введите конец интервала: ");

check\_segment(a, b);

const double h = get\_value("Введите шаг функции: ");

check\_step(h);

double x = a;

const double e = get\_value("Введите точность вычислений: ");

check\_epsilon(e, x);

while (x - b <= DBL\_EPSILON)

{

printf\_s("%10.2lf | %25.15lf | %.15lf \n", x, get\_function(x), get\_series\_sum(x, e));

x += h;

}

return 0;

}

double get\_series\_sum(double x, double e)

{

double current = 1;

double sum = current;

double k = 0;

while (get\_function(x) - sum > e + DBL\_EPSILON)

{

current \*= get\_recurrent(x, k);

sum += current;

k++;

}

return sum;

}

double get\_recurrent(double x, double k)

{

return pow(x, 2) / ((2 \* k + 2) \* (2 \* k + 1));

}

double get\_function(double x)

{

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

}

void check\_step(double h)

{

if (h < DBL\_EPSILON)

{

puts("Неверно введено значение!");

abort();

}

}

void check\_segment(const double a, const double b)

{

if (a - b > -DBL\_EPSILON)

{

puts("Неверно введен интервал.");

abort();

}

}

double get\_value(const char\* message)

{

double value;

printf("%s", message);

int result = scanf\_s("%lf", &value);

if (result != 1)

{

errno = EIO;

perror("Ошибка ввода");

abort();

}

return value;

}

void check\_epsilon(double e, double x)

{

if (e < DBL\_EPSILON || e - get\_first\_element(x) > DBL\_EPSILON)

{

printf("Неверно введена тосность!");

abort();

}

}

double get\_first\_element(double x)

{

double first = pow(x, 2) / 2;

return first;

}

* 1. Решение тестового примера



Рисунок 37 – Решение тестового примера



Рисунок 38 – Вывод программы, когда начало отрезка – буква

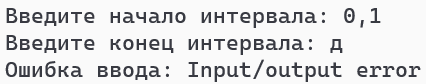


Рисунок 39 – Вывод программы, когда конец отрезка – буква

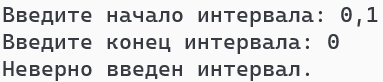


Рисунок 40 – Вывод программы, когда начало отрезка больше конца

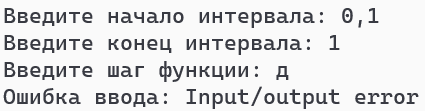


Рисунок 41 – Вывод программы, когда шаг функции – буква

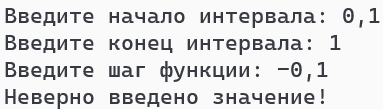


Рисунок 42 – Вывод программы, когда шаг функции отрицательный

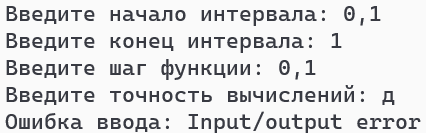


Рисунок 43 – Вывод программы, когда точность вычислений – буква

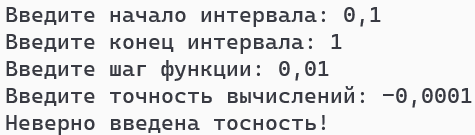


Рисунок 44 – Вывод программы, когда точность вычислений отрицательная

* 1. Решение тестовых примеров в Excel

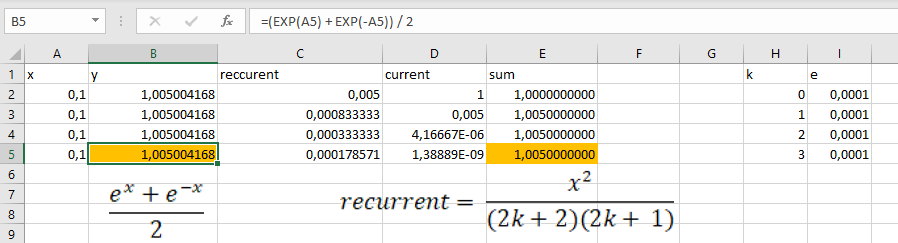


Рисунок 45 – Расчёт функции по формуле в Excel

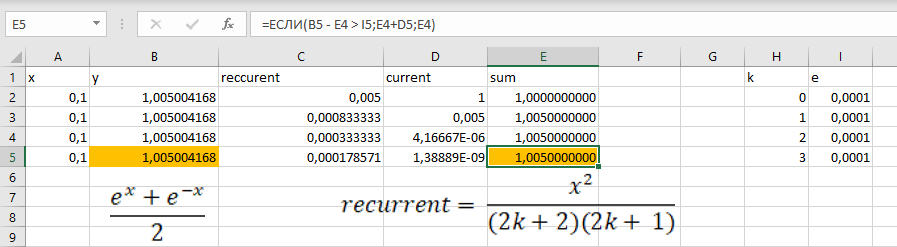


Рисунок 46 – Расчёт суммы функционального ряда с точностью e по рекуррентной формуле в Excel

* 1. Зачёт задания в GitHub

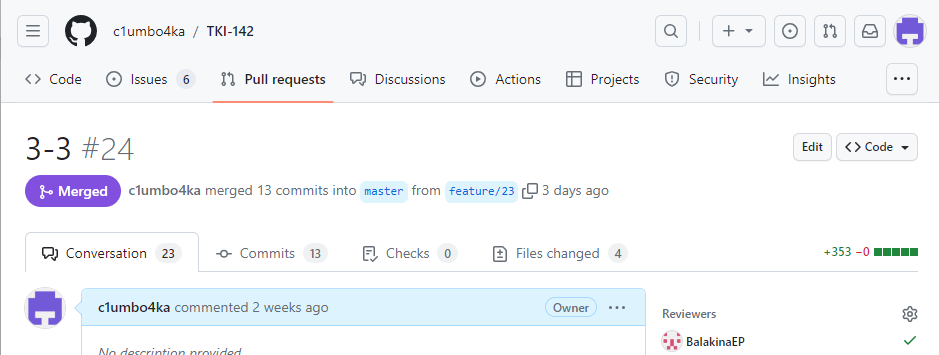


Рисунок 47 – Зачёт задания в GitHub