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 5

Выполнил: ст. гр. ТКИ-142

Гаспарян Артём Артурович

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

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

Москва 2024

Оглавление

[1 ЗАДАНИЕ 3.1 4](#_Toc184763900)

[1.1 Формулировка задания 4](#_Toc184763901)

[1.2 Блок-схема алгоритма 5](#_Toc184763902)

[1.3 Текст программы на языке С 10](#_Toc184763903)

[1.4 Результат выполнения программы 14](#_Toc184763904)

[1.5 Выполнение тестовых примеров 16](#_Toc184763905)

[1.6 Отметка о выполнение задания 17](#_Toc184763906)

[2 ЗАДАНИЕ 3.2 18](#_Toc184763907)

[2.1 Формулировка задания 18](#_Toc184763908)

[2.2 Блок-схема алгоритма 19](#_Toc184763909)

[2.3 Текст программы на языке С 24](#_Toc184763910)

[2.4 Результат выполнения программы 28](#_Toc184763911)

[2.5 Выполнение тестовых примеров 30](#_Toc184763912)

[2.6 Отметка о выполнение задания 32](#_Toc184763913)

[3 ЗАДАНИЕ 3.3 33](#_Toc184763914)

[3.1 Формулировка задания 33](#_Toc184763915)

[3.2 Блок-схема основного алгоритма 34](#_Toc184763916)

[3.3 Код программы на языке С 38](#_Toc184763917)

[3.4 Результат выполнения программы 41](#_Toc184763918)

[3.5 Выполнение тестовых примеров 43](#_Toc184763919)

[3.6 Отметка о выполнение задания 44](#_Toc184763920)

1. задание 3.1

1.1 Формулировка задания

Протабулировать заданную в таблице функцию (Таблица 1). 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  – Исходные данные

|  |  |  |
| --- | --- | --- |
| **Вариант** | **Функция** | **Константы** |
| **5** |  |  |

1.2 Блок-схема алгоритма

Блок-схема основного алгоритма представлена ниже (Рисунок 1). Блок-схемы функций расчета (Рисунок 2, Рисунок 3, Рисунок 4, Рисунок 5, Рисунок 6)

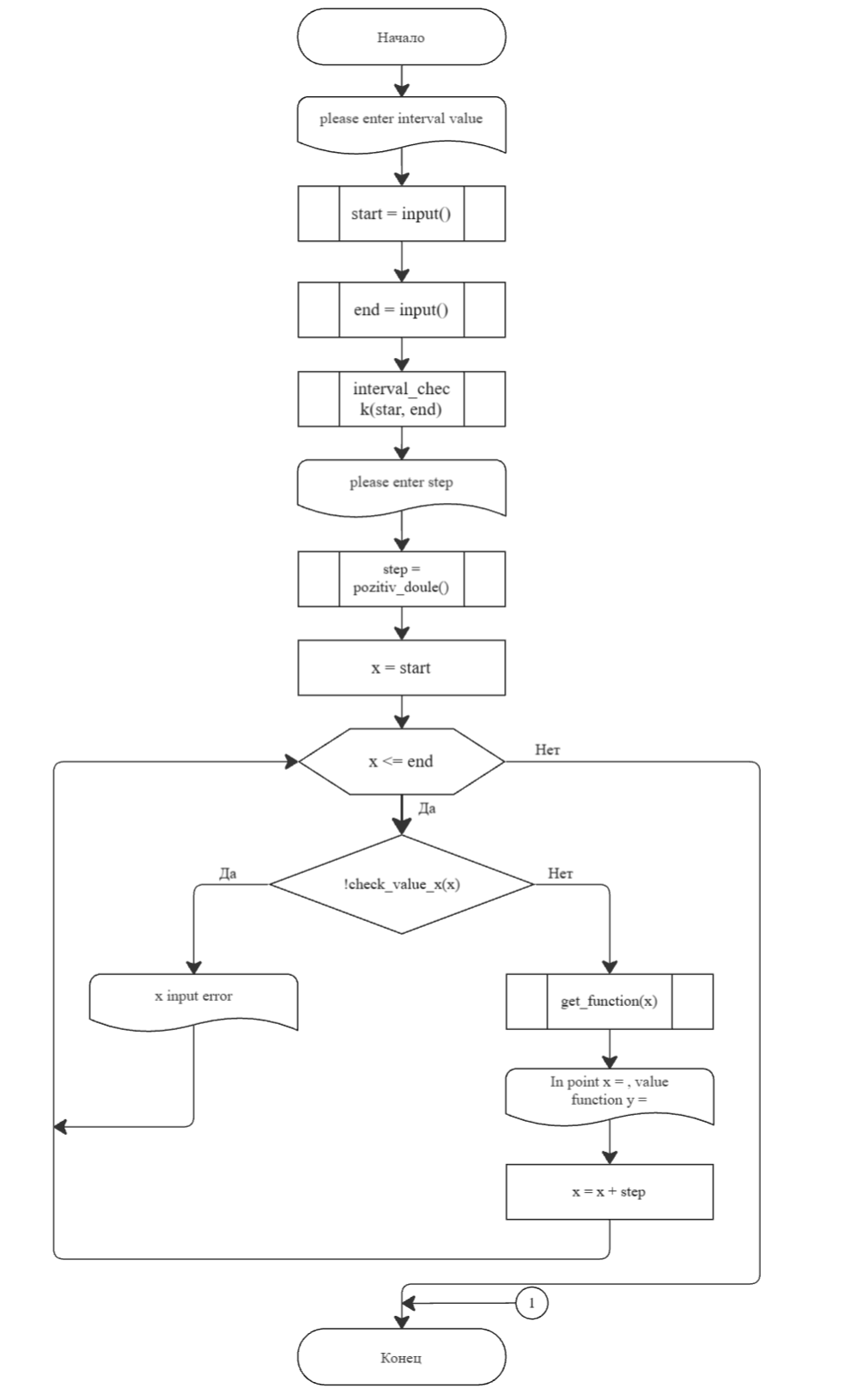


Рисунок  – Блок-схема основного алгоритма

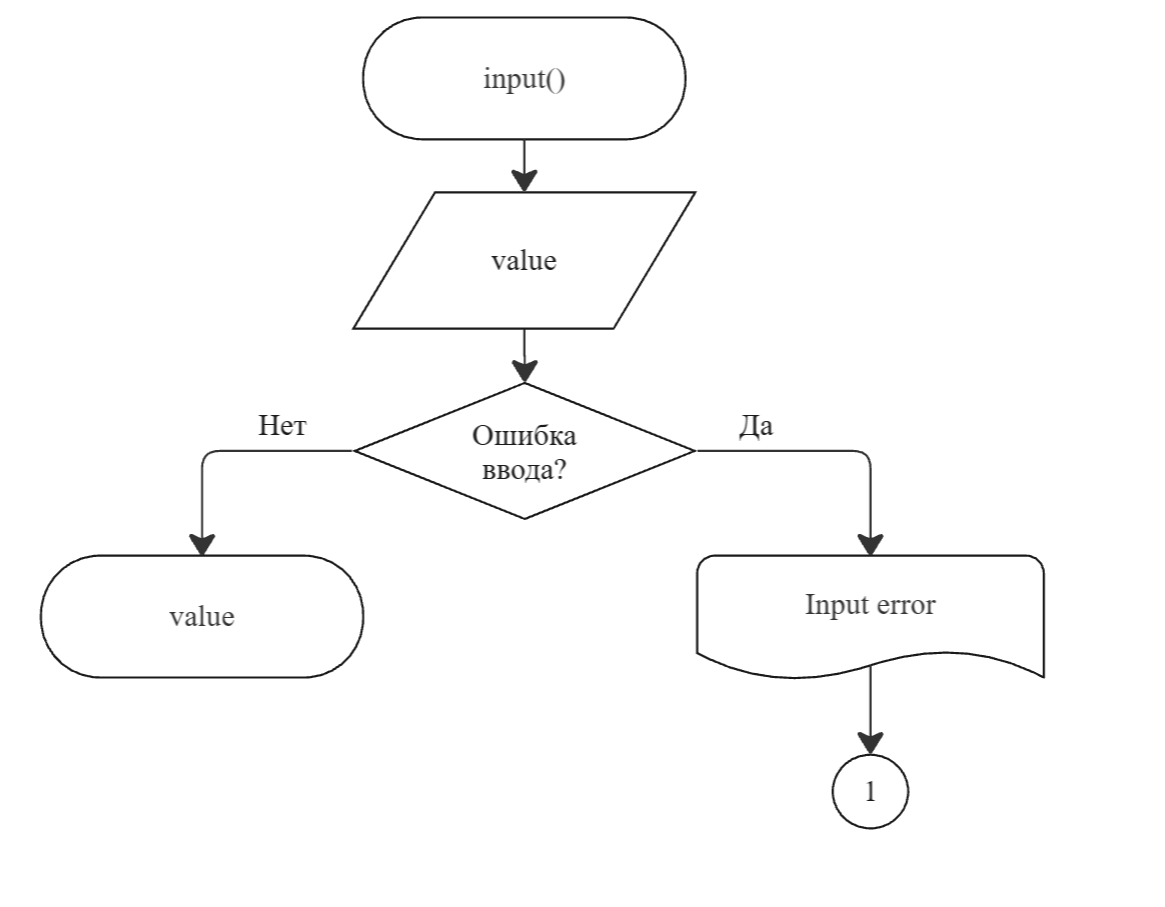


Рисунок  – Блок-схема функции input

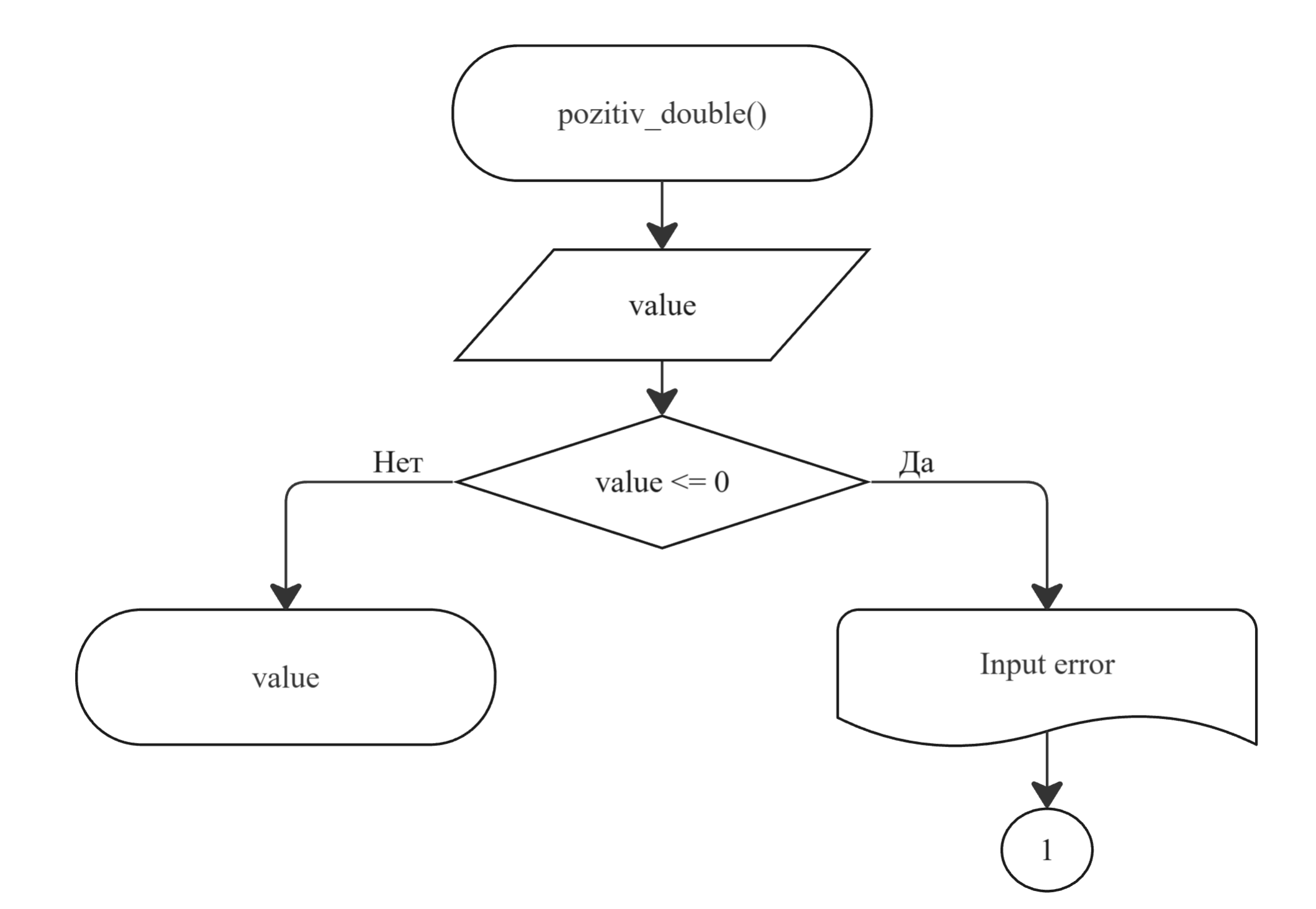


Рисунок  – Блок-схема функции pozitiv\_double

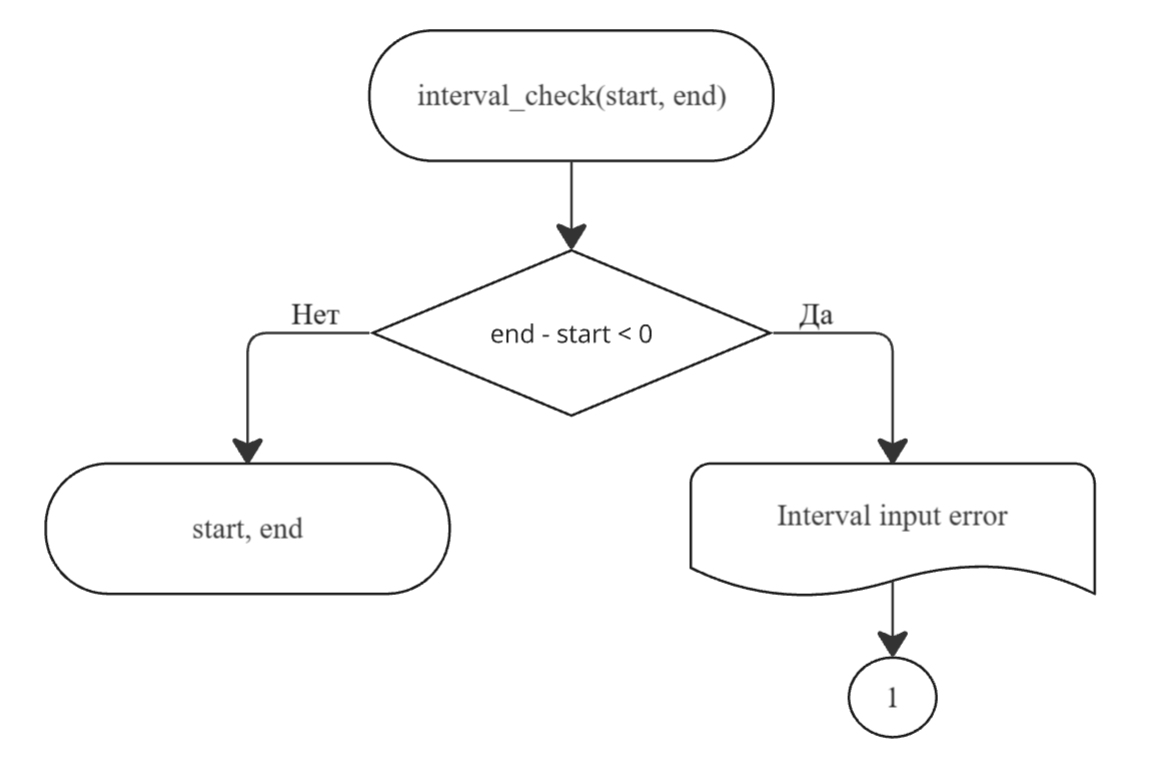


Рисунок  – Блок-схема функции interval\_check

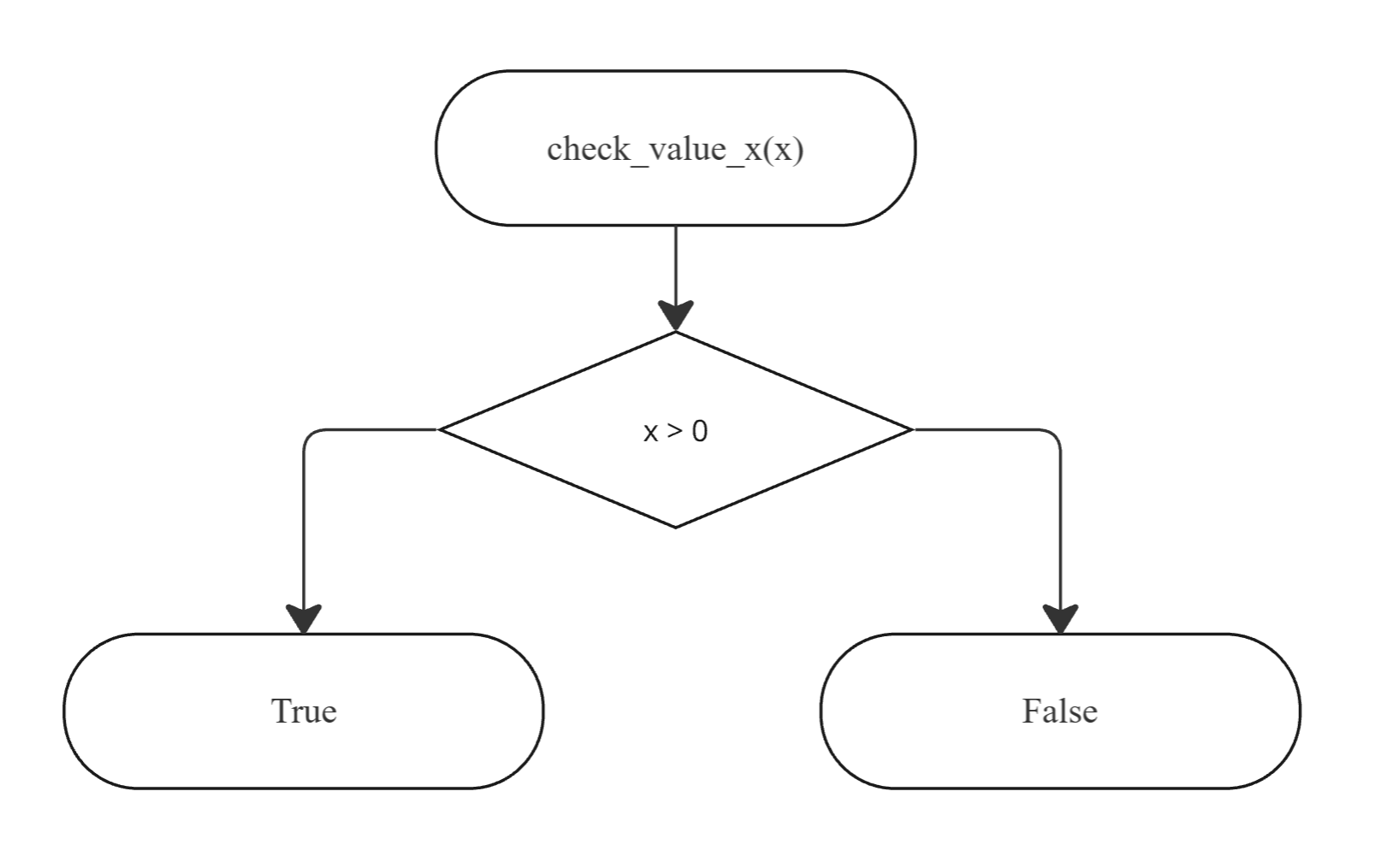


Рисунок 5 – Блок-схема функции check\_value\_x(x)

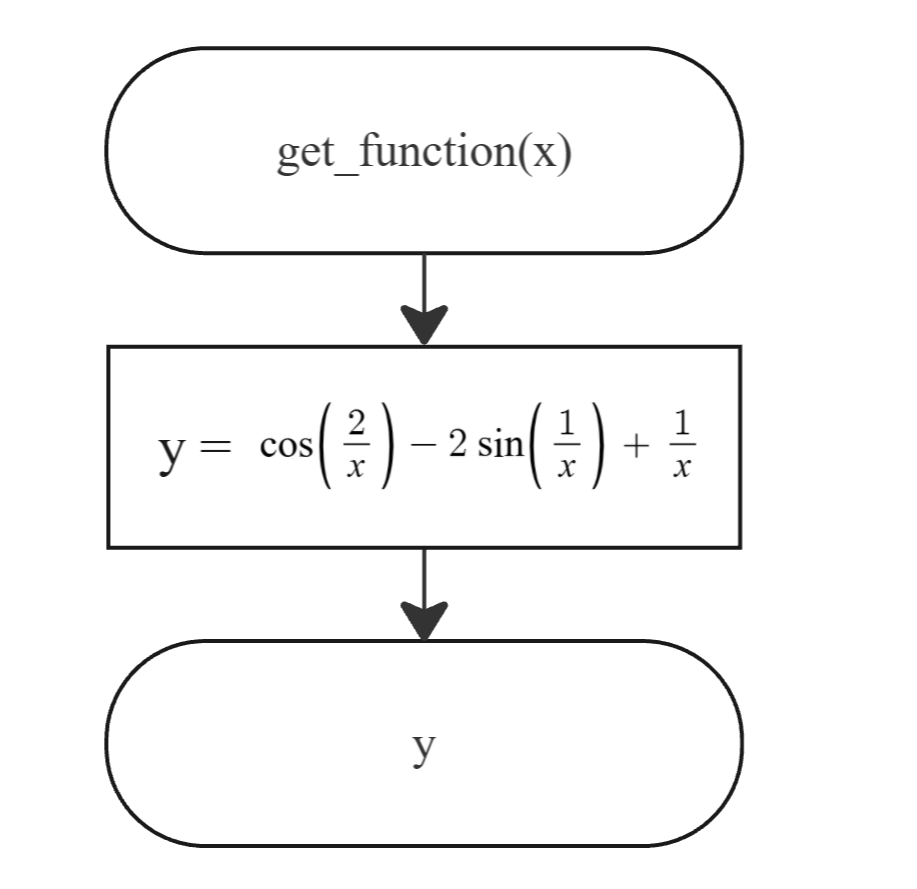


Рисунок  – Блок-схема функции get\_function(x)

1.3 Текст программы на языке С

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <stdbool.h>

#include <errno.h>

#include <float.h>

/\*\*

\* @brief получает на вход значение

\* @return возвращает полученное значение, в противном случае ошибку

\*/

double input(void);

/\*\*

\* @brief проверяет заданное число на положительность

\* @return возвращает проверенное число, в ином случаешь завершает программу и пишет ошибку

\*/

double pozitiv\_double(void);

/\*\*

\* @brief проверяет корретность заданного интервала

\* @param start начальное значение заданного интервала

\* @param end конечное значение заданного интервала

\* @return в случае некорректно заданного интервала, возвращает ошибку

\*/

void interval\_check(const double start, const double end);

/\*\*

\* @brief проверяет значение заданного параметра x

\* @param x значение параметра х

\* @return возвращает True, если все введенно корректно, в противном случае вернет False

\*/

bool check\_value\_x(const double x);

/\*\*

\* @brief рассчитывает значение заданной фукнции

\* @param x значение переменной x

\* @return рассчитанное значение функции

\*/

double get\_function(const double x);

/\*\*

\* @brief точка входа в программу

\* @return 0 в случае успеха

\*/

int main(void)

{

printf("please enter interval value\n");

const double start = input();

const double end = input();

interval\_check(start, end);

printf("please enter step\n");

const double step = pozitiv\_double();

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

{

if (!check\_value\_x(x))

{

puts("x input error");

}

else {

printf("In point x = %.3lf, value function y = %.3lf\n", x, get\_function(x));

}

}

return 0;

}

double input(void)

{

double value = 0.0;

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

if (result != 1)

{

errno = EIO;

perror("Input error value\n");

exit(EXIT\_FAILURE);

}

return value;

}

double pozitiv\_double(void)

{

double value = input();

if (value < DBL\_EPSILON)

{

errno = EIO;

perror("Input error\n");

exit(EXIT\_FAILURE);

}

return value;

}

void interval\_check(const double start, const double end)

{

if (end - start < DBL\_EPSILON)

{

errno = EIO;

perror("Interval input error\n");

exit(EXIT\_FAILURE);

}

}

void step\_check(const double step)

{

if (step <= DBL\_EPSILON)

{

errno = EIO;

perror("Step input error\n");

exit(EXIT\_FAILURE);

}

}

bool check\_value\_x(const double x)

{

return x > DBL\_EPSILON;

}

double get\_function(const double x)

{

return cos(2 / x) - 2 \* sin(1 / x) + 1 / x;

}

1.4 Результат выполнения программы

Результаты выполнения программы представлены ниже (Рисунок 7, Рисунок 8, Рисунок 9, Рисунок 10, Рисунок 11, Рисунок 12).

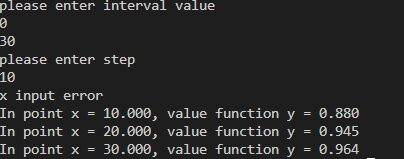


Рисунок  – Первый результат выполнения программы

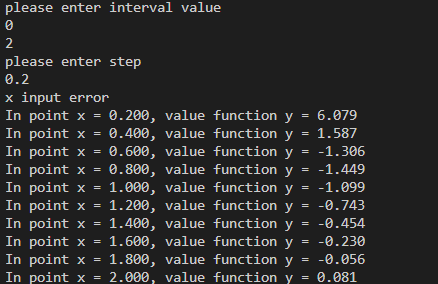


Рисунок  – Второй результат выполнения программы

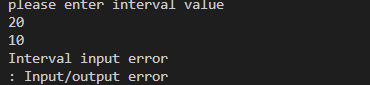


Рисунок  – Проверка функции interval\_check

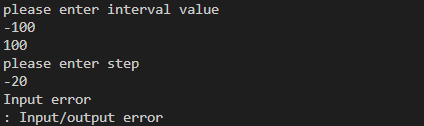


Рисунок  – Проверка функции pozitiv\_double

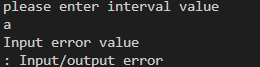


Рисунок  – Проверка функции input

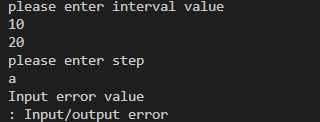


Рисунок  – Проверка функции input, если step нечисловое значени

1.5 Выполнение тестовых примеров

В программе MS Excel были выполнены тестовые примеры (Рисунок 13, Рисунок 14).

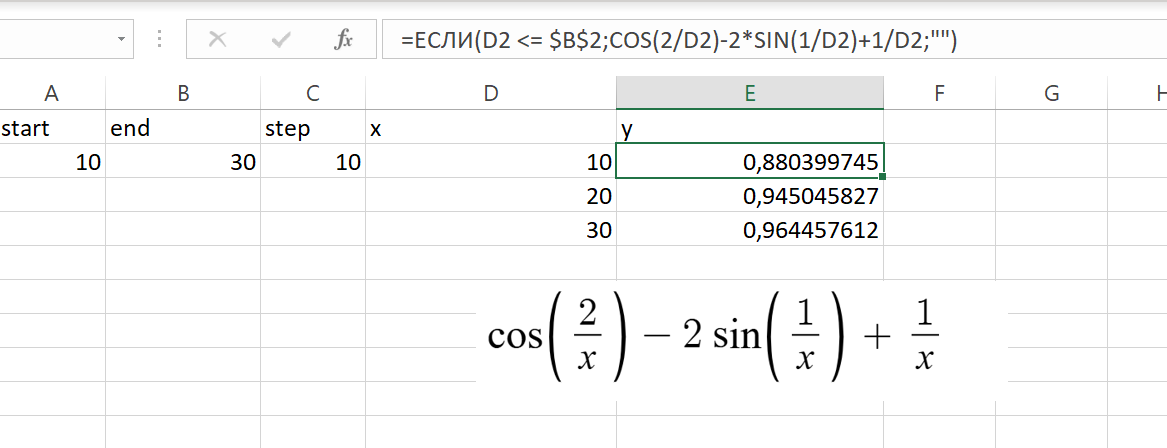


Рисунок  – Результат выполнения первого тестового примера

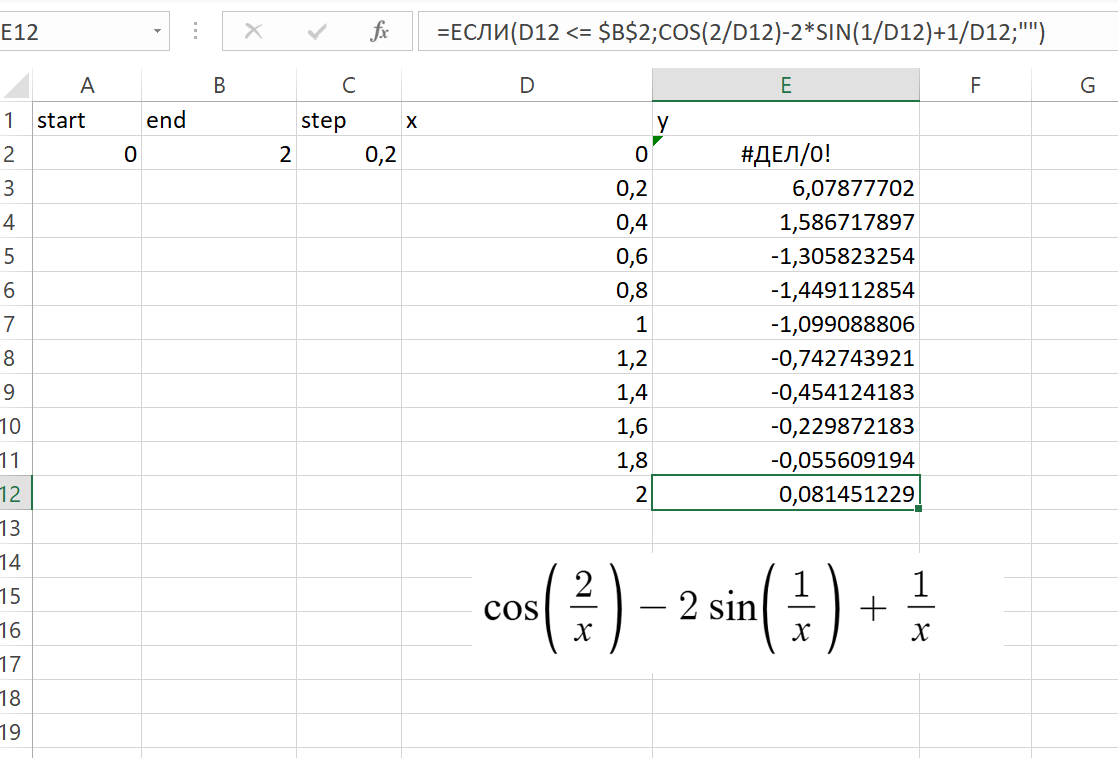


Рисунок  – Результат выполнения второго тестового примера

1.6 Отметка о выполнение задания

Отметка о выполнение задания в GitHub

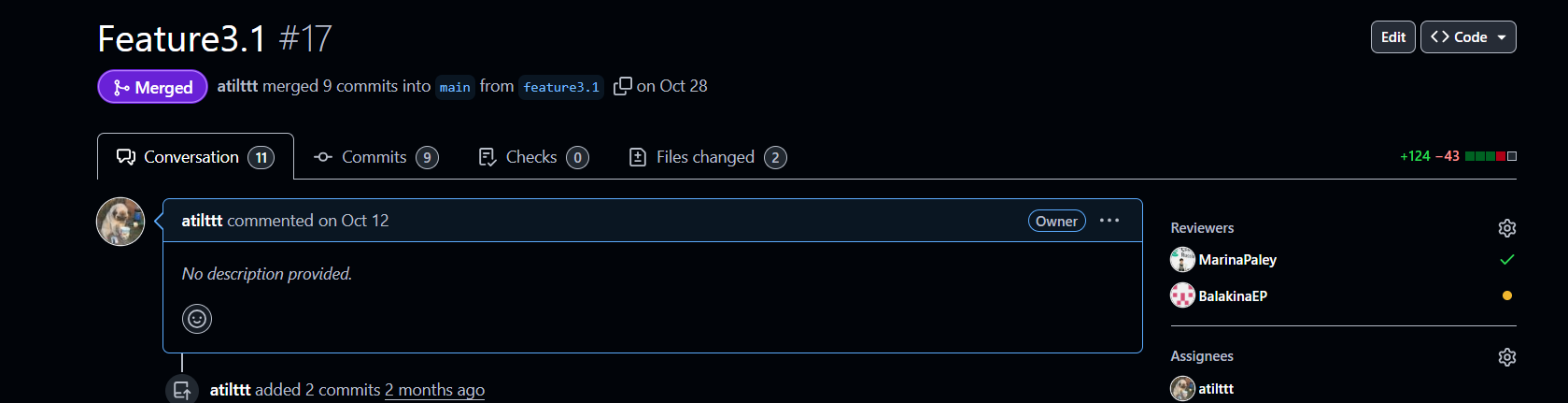


Рисунок  – Отметка о выполнение задания

1. ЗАДАНИ 3.2

2.1 Формулировка задания

Составьте две программы (Таблица 2):

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

Помните о проверке пользовательского ввода. Все результаты вывести на экран. Отчёт дополнить блок-схемой. При вычислении факториалов рекомендуется отказаться от использования рекурсивных методов.

Таблица – Исходные данные

|  |  |
| --- | --- |
| **Вариант** | **Ряд** |
| **5** |  |

2.2 Блок-схема алгоритма

Блок-схема основного алгоритма представлена ниже (Рисунок 16). Блок-схемы функций расчета (Рисунок 17, Рисунок 18, Рисунок 19, Рисунок 20, Рисунок 21).

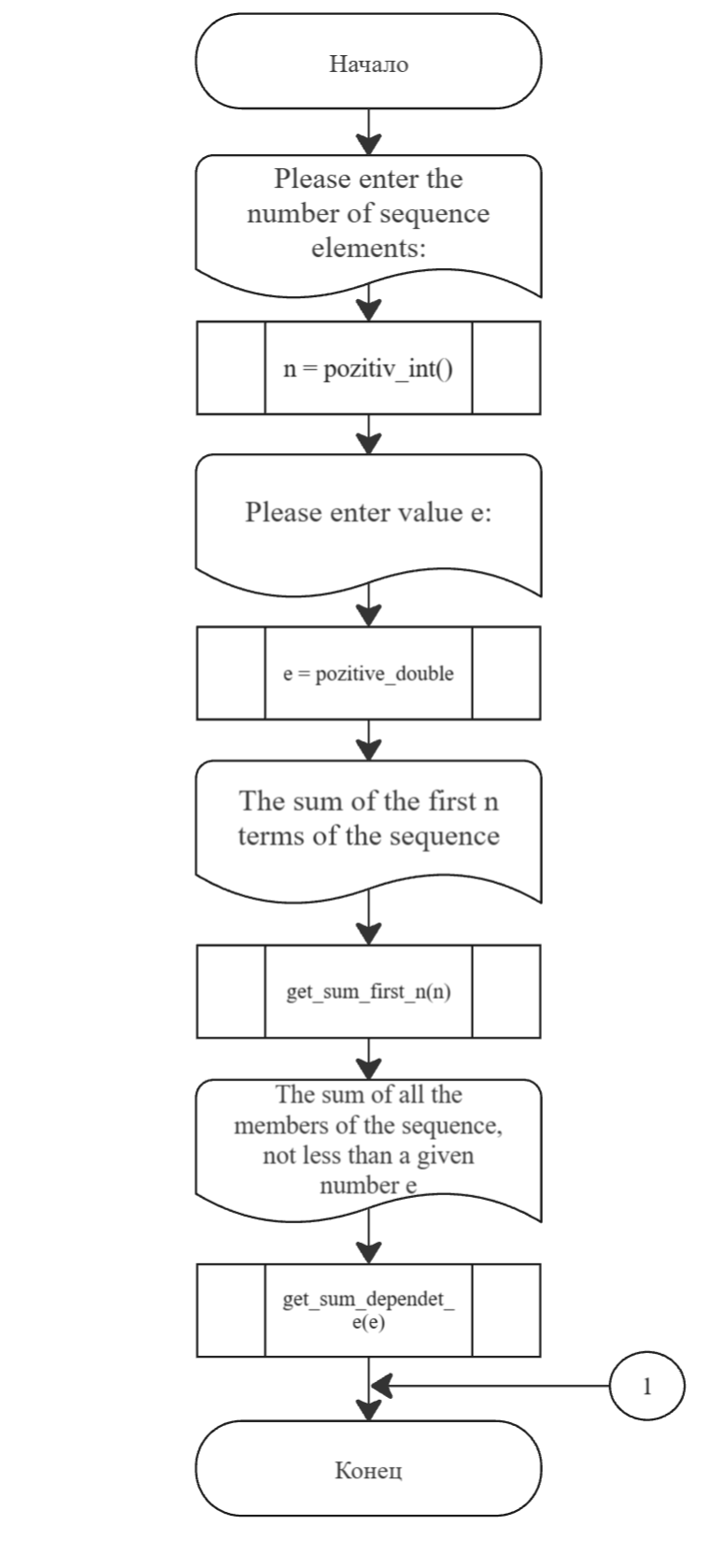


Рисунок  – Блок-схема основного алгоритма

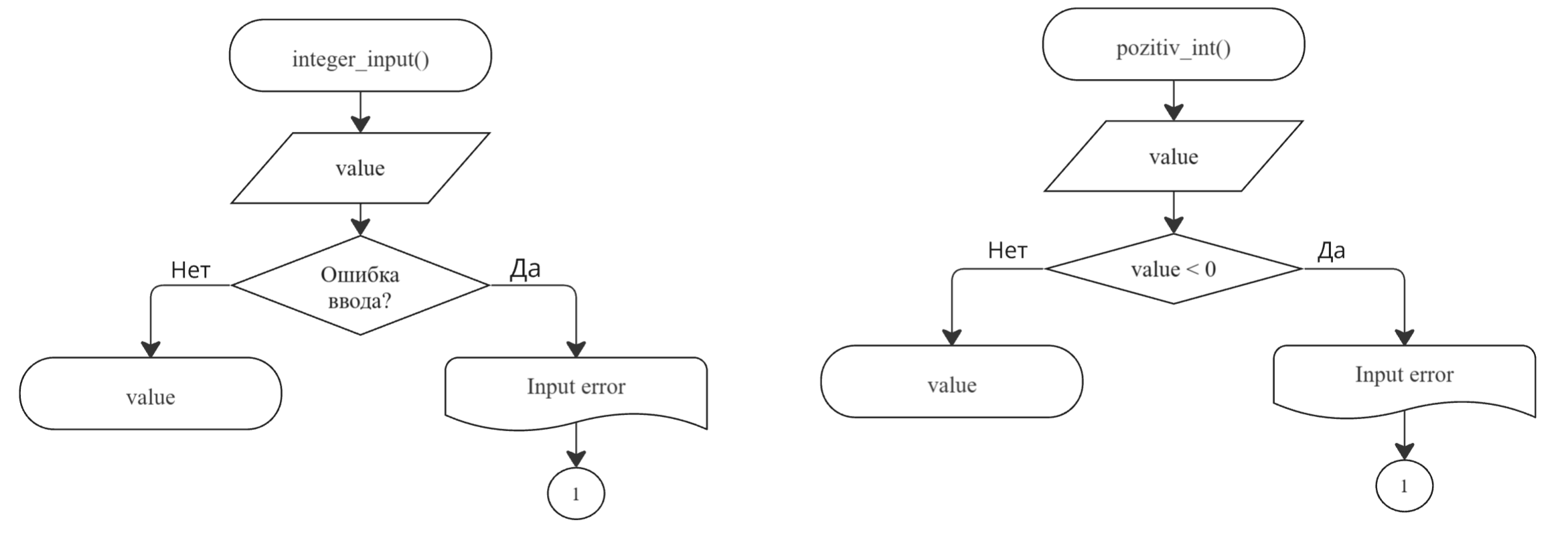


Рисунок  – Блок-схемы функций integer\_input и pozitiv\_int

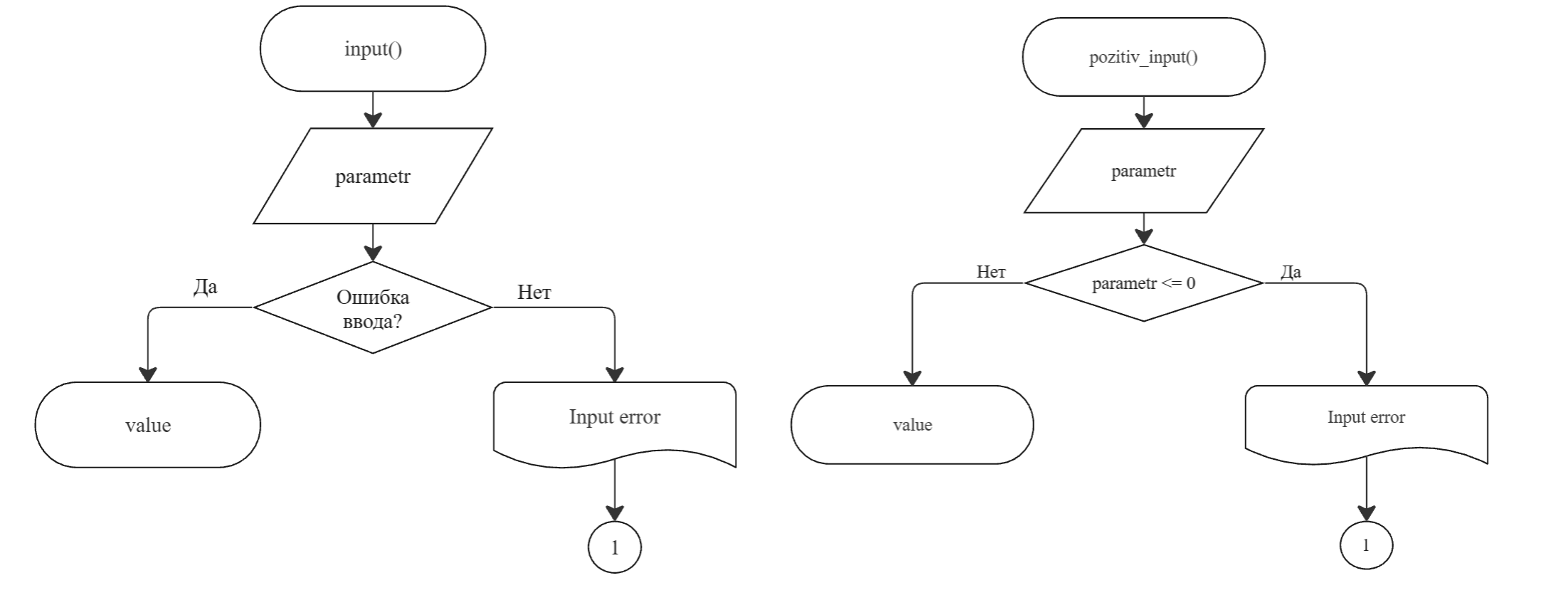


Рисунок  – Блок-схемы функций pozitiv\_input и input

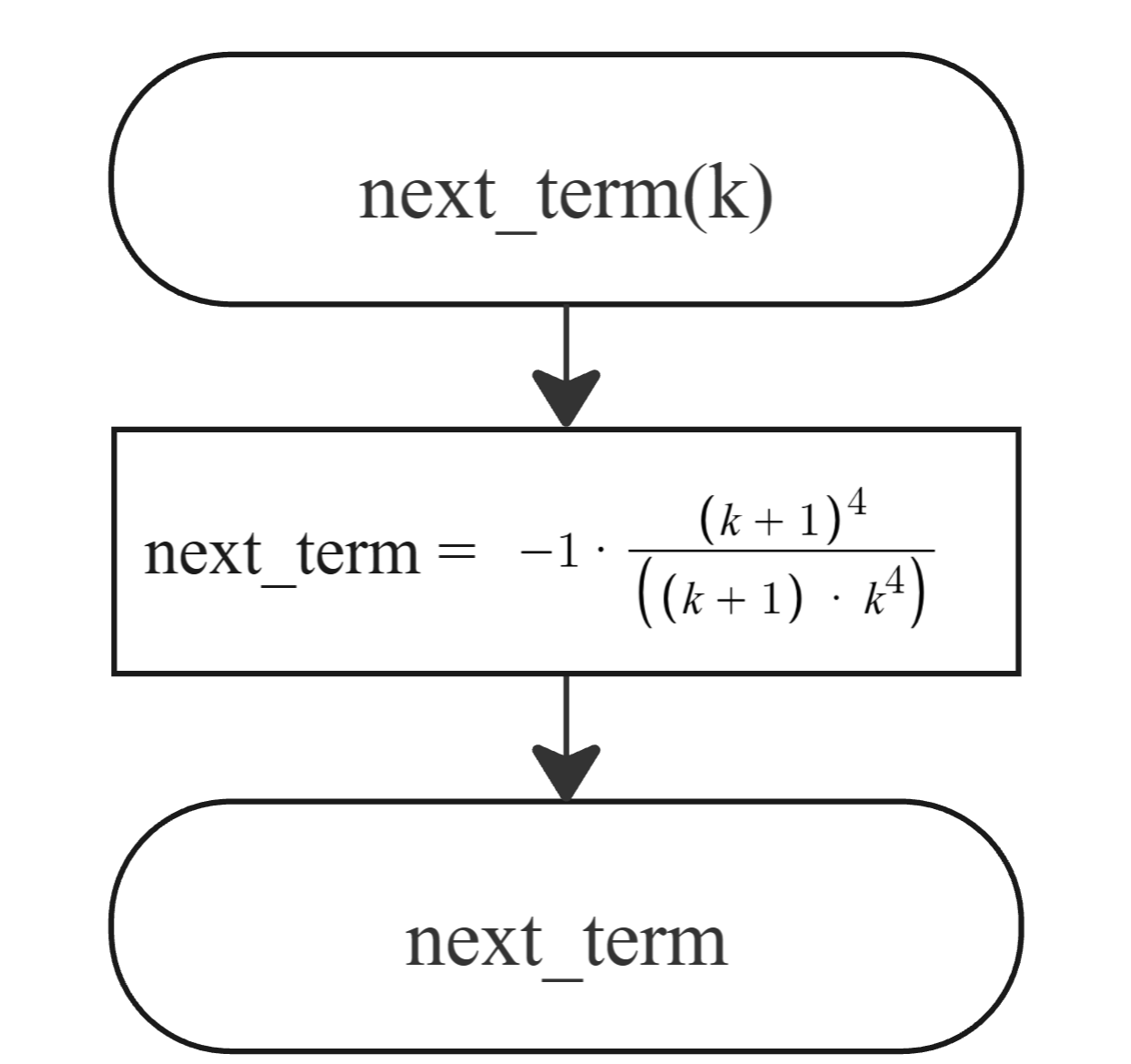


Рисунок  – Блок-схема функции next\_term(k)

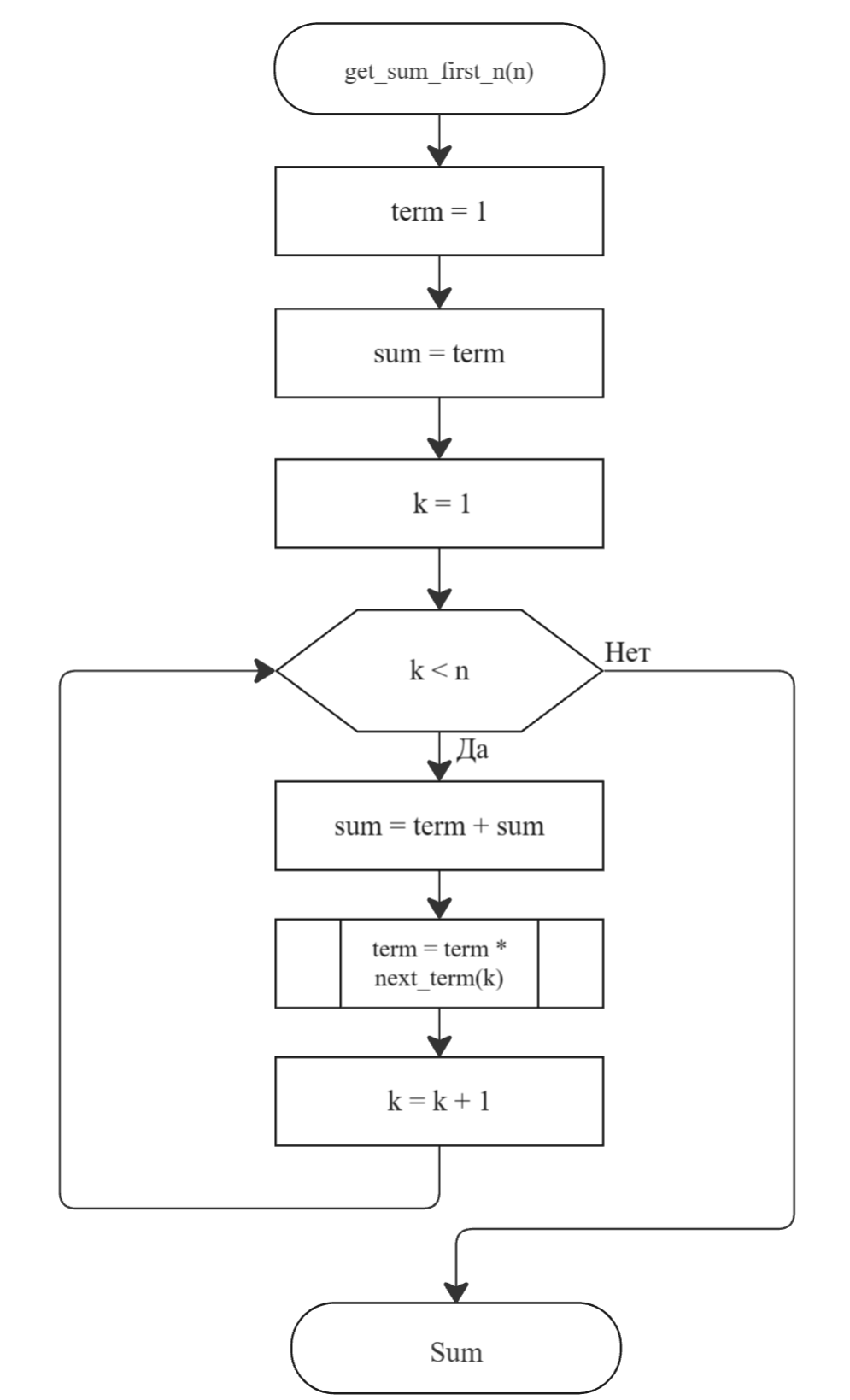


Рисунок  – Блок-схема функции get\_sum\_first\_n(n)

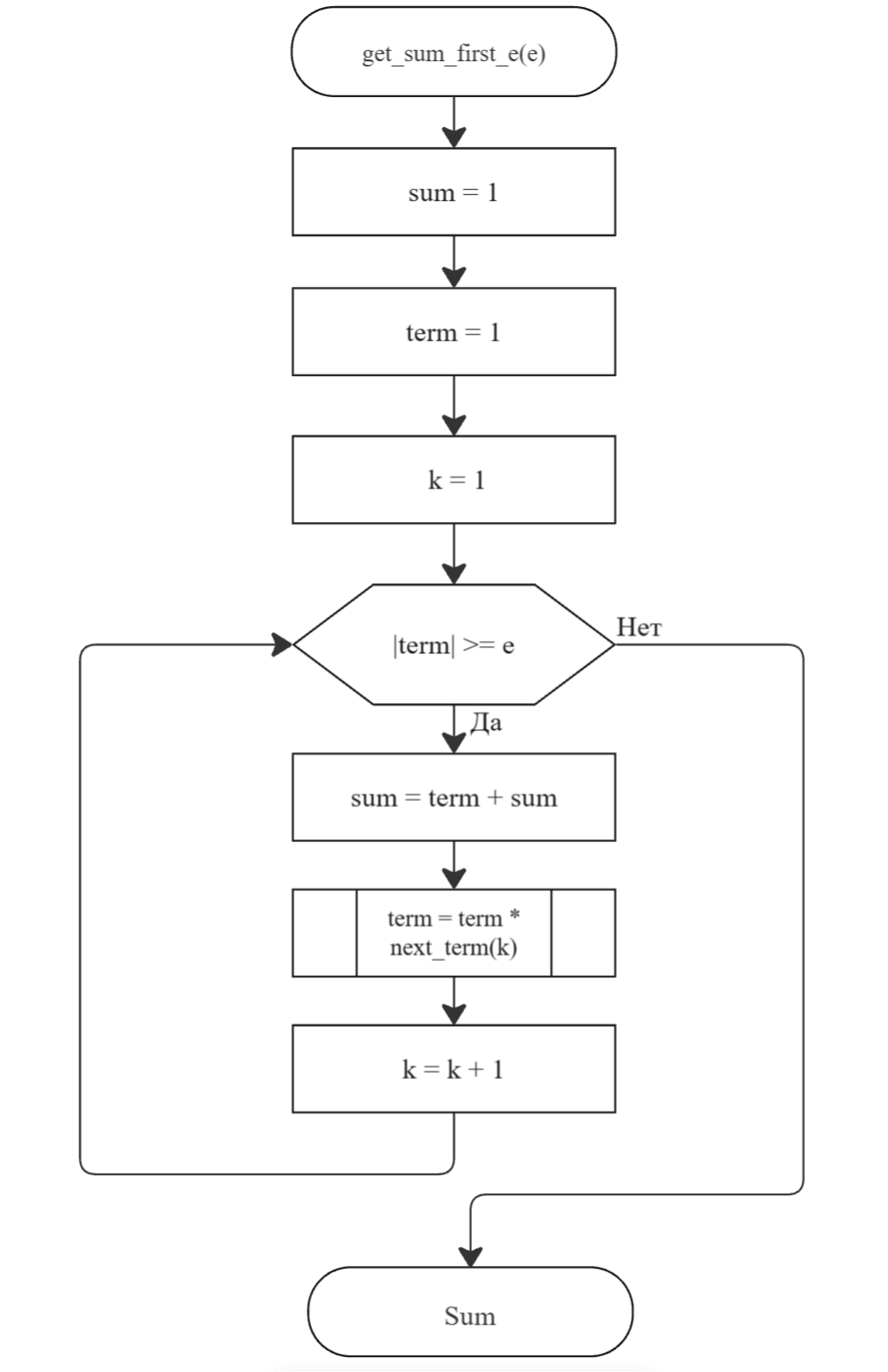


Рисунок  – Блок-схема функции get\_sum\_dependet\_e(e)

2.3 Текст программы на языке С

#include <stdio.h>

#include <math.h>

#include <errno.h>

#include <stdlib.h>

#include <float.h>

/\*\*

\* @brief получает на вход число из потока ввода

\* @return возвращает проверенное число, в противном случае ошибку

\*/

double input(void);

/\*\*

\* @brief получает на вход число из потока ввода

\* @return возвращает провернное число, в противном случае ошибку

\*/

int integer\_input(void);

/\*\*

\* @brief проверяет заданное число на положительность

\* @return возвращает проверенное число, в ином случае завершает программу и пишет ошибку

\*/

int positive\_int(void);

/\*\*

\* @brief проверяет заданное число на положительность

\* @return возвращает проверенное число, в ином случае завершает программу и пишет ошибку

\*/

double pozitive\_double(void);

/\*\*

\* @brief вычисляет следующий член последовательности при помощи рекуррентного выражения

\* @param k текущий индекс последовательности

\* @return значение следующего члена последовательности

\*/

double next\_term(const int k);

/\*\*

\* @brief рассчитывает значение суммы первых n-членов последовательности

\* @param n кол-во элементов последовательности

\* @return рассчитанное значение суммы

\*/

double get\_sum\_first\_n(const int n);

/\*\*

\* @brief рассчитывает значение суммы всех членов последовательности, не меньше заданного числа е

\* @param n кол-во элементов последовательности

\* @param e значение заданного числа е

\* @return рассчитанное значение суммы

\*/

double get\_sum\_dependet\_e(const double e);

/\*\*

\* @brief точка входа в программу

\* @return 0 в случае успеха

\*/

int main(void)

{

printf("Please enter the number of sequence elements:\n");

const int n = positive\_int();

printf("Please enter value e:\n");

const double e = pozitive\_double();

printf("The sum of the first n terms of the sequence = %.3lf\n", get\_sum\_first\_n(n));

printf("The sum of all the members of the sequence, not less than a given number e = %.3lf\n", get\_sum\_dependet\_e(e));

return 0;

}

int integer\_input(void)

{

int value = 0;

int result = scanf("%d", &value);

if (result != 1) {

errno = EIO;

perror("Input error");

exit(EXIT\_FAILURE);

}

return value;

}

int positive\_int(void)

{

int value = integer\_input();

if (value < 0)

{

errno = EIO;

perror("Impossible value for n");

exit(EXIT\_FAILURE);

}

return value;

}

double input(void)

{

double parameter = 0.0;

int outcome = scanf("%lf", &parameter);

if (outcome != 1) {

errno = EIO;

perror("Input error\n");

exit(EXIT\_FAILURE);

}

return parameter;

}

double pozitiv\_double(void)

{

double parameter = input();

if (parameter <= DBL\_EPSILON)

{

errno = EIO;

perror("Input error\n");

exit(EXIT\_FAILURE);

}

return parameter;

}

double next\_term(const int k) {

return -1 \* (pow(k, 4) / k);

}

double get\_sum\_first\_n(const int n)

{

double term = 1.0;

double sum = term; // Начальный член a\_1 = 1^4 / 1!

for (int k = 1; k < n; ++k) {

term \*= next\_term(k);

sum += term;

}

return sum;

}

double get\_sum\_dependet\_e(const double e)

{

double sum = 1.0;

double term = 1.0; // Начальный член a\_1 = 1^4 / 1!

int k = 1;

while (fabs(term) >= e + DBL\_EPSILON)

{

sum += term;

term \*= next\_term(k);

k++;

}

return sum;

}

2.4 Результат выполнения программы

Результат выполнения программы представлен ниже (Рисунок 22, Рисунок 23, Рисунок 24, Рисунок 25, Рисунок 26, Рисунок 27)

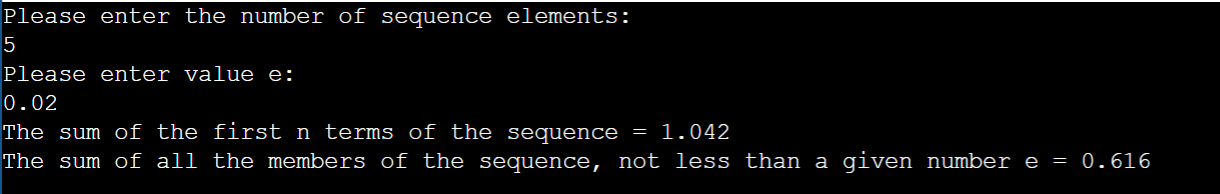


Рисунок  – Первый результат выполнения программы

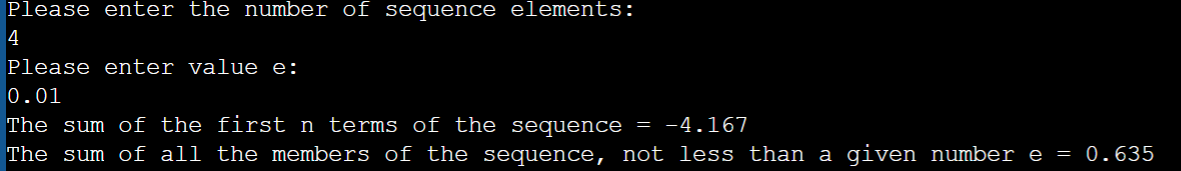


Рисунок  – Второй результат выполнения программы

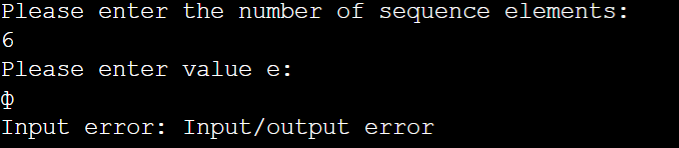


Рисунок  – Проверка функции input()

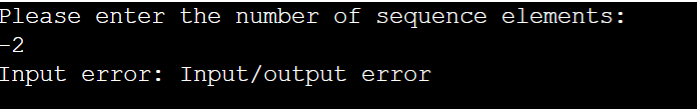


Рисунок  – Проверка функции pozitiv\_int

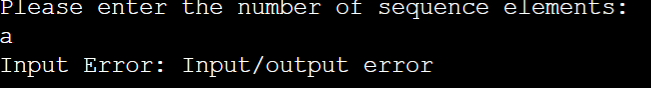


Рисунок  – Проверка функции integer\_input()

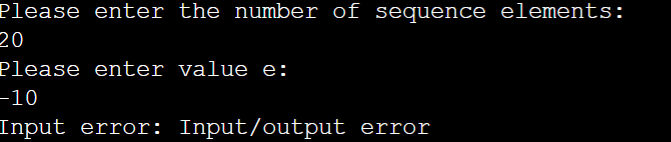


Рисунок  – Проверка функции pozitiv\_double()

2.5 Выполнение тестовых примеров

В программе MS Excel были выполнены тестовые примеры. Результат их выполнения представлен ниже (Рисунок 28, Рисунок 29).

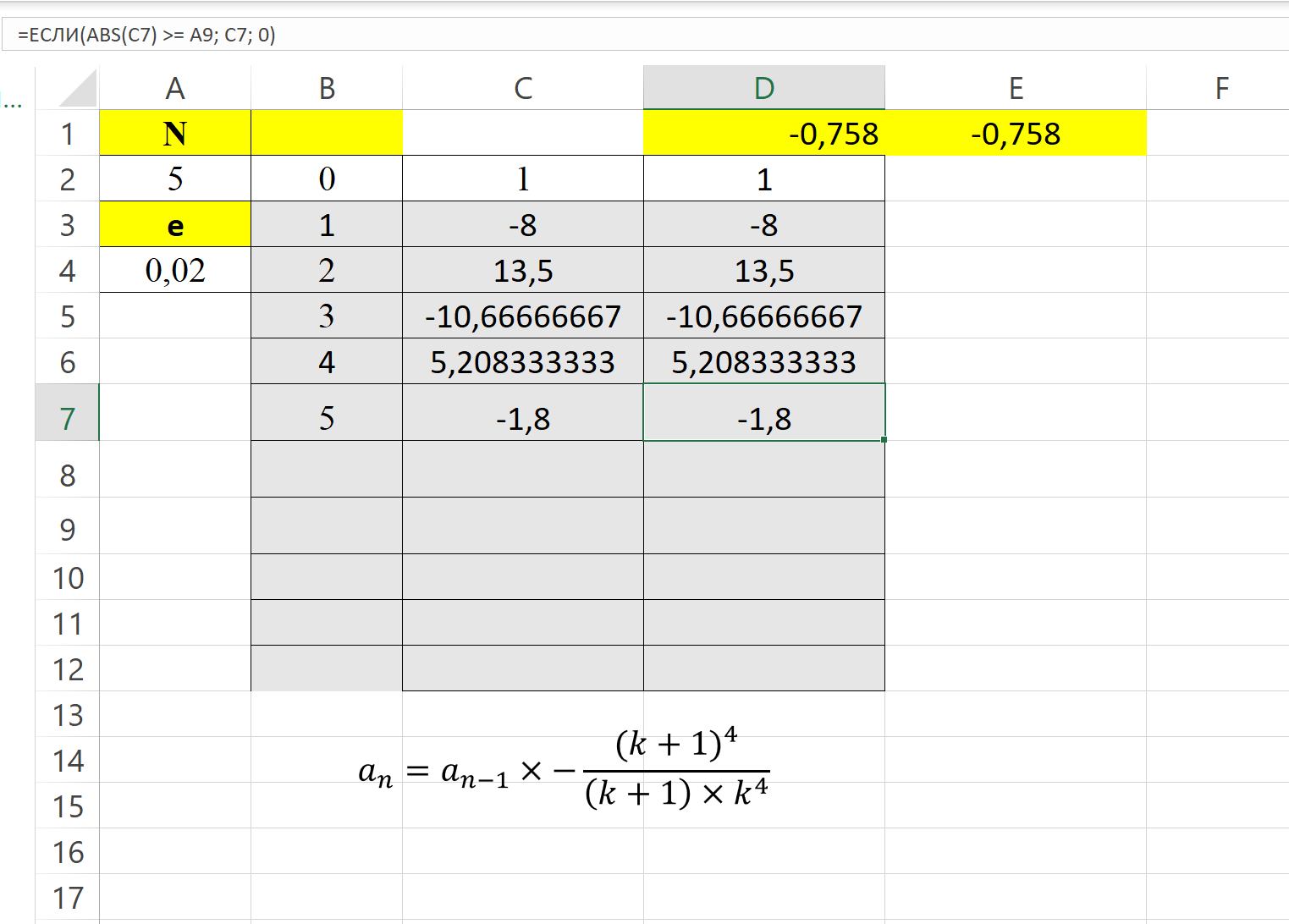


Рисунок  – Выполнение первого тестового примера

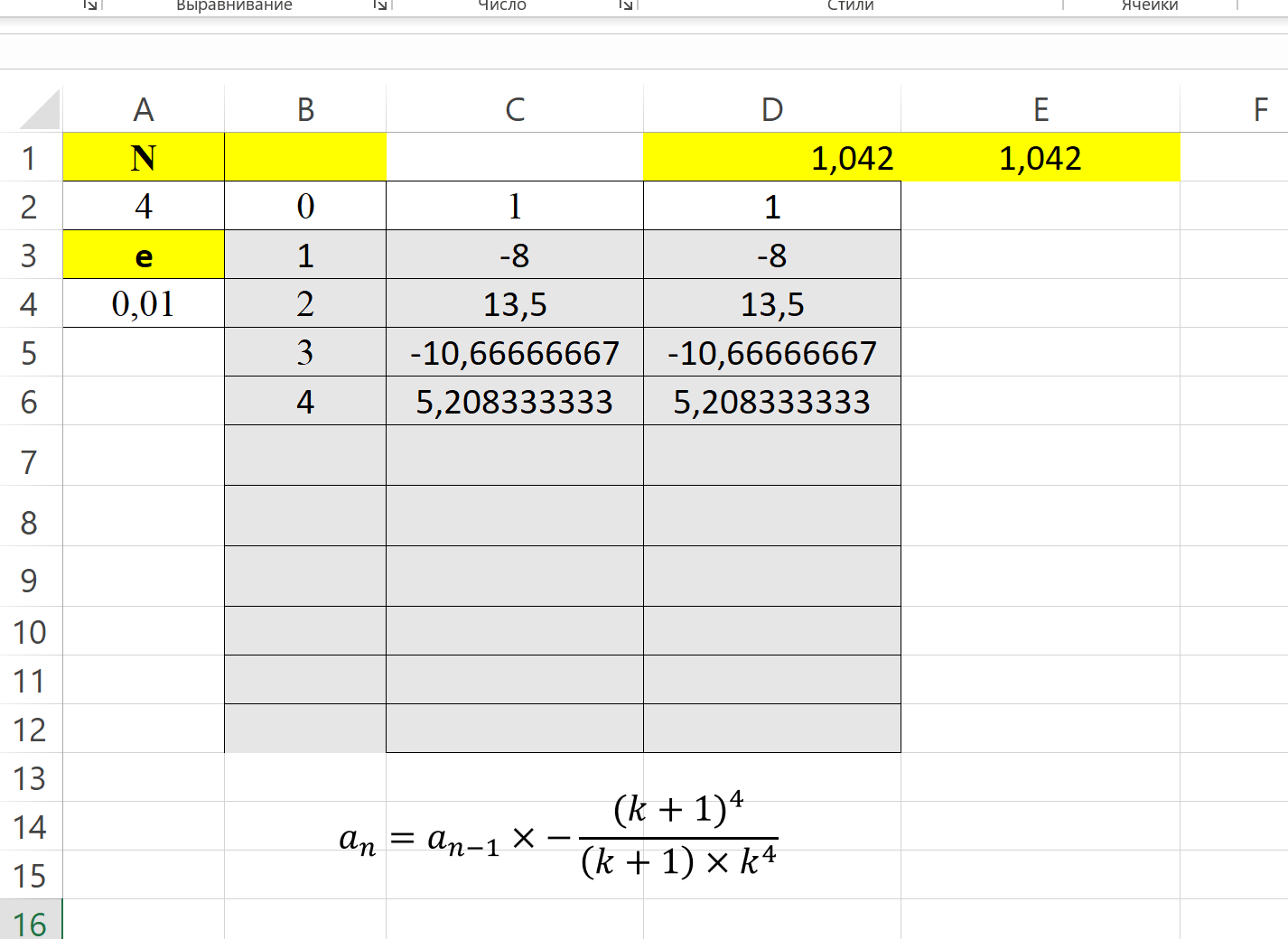


Рисунок  – Выполнение 2-ого тестового примера

2.6 Отметка о выполнение задания

Отметка о выполнение задания в GitHub (Рисунок 30).

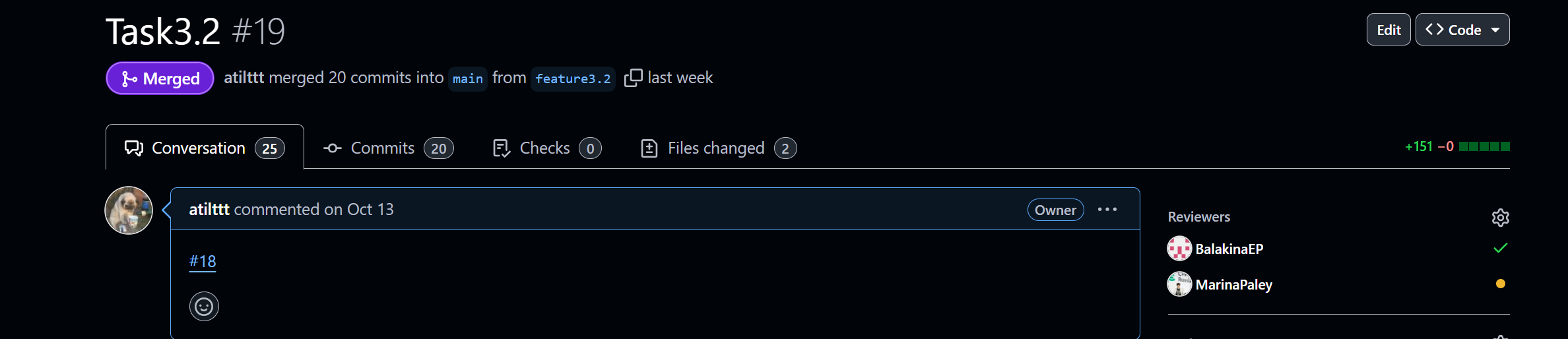


Рисунок  – Отметка о выполнение задания

1. ЗАДАНИЕ 3.3

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). Функциональный 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).

Таблица  – Исходные данные

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Вариант** | **Функция y** | **Сумма** | **Интервал** | **ɛ** |
| 5 |  |  |  | **30-5** |

3.2 Блок-схема основного алгоритма

Блок-схема основного алгоритма представлена ниже (Рисунок 31). Блок-схемы функций расчета (Рисунок 32, Рисунок 33, Рисунок 34, Рисунок 35, Рисунок 36, Рисунок 37).

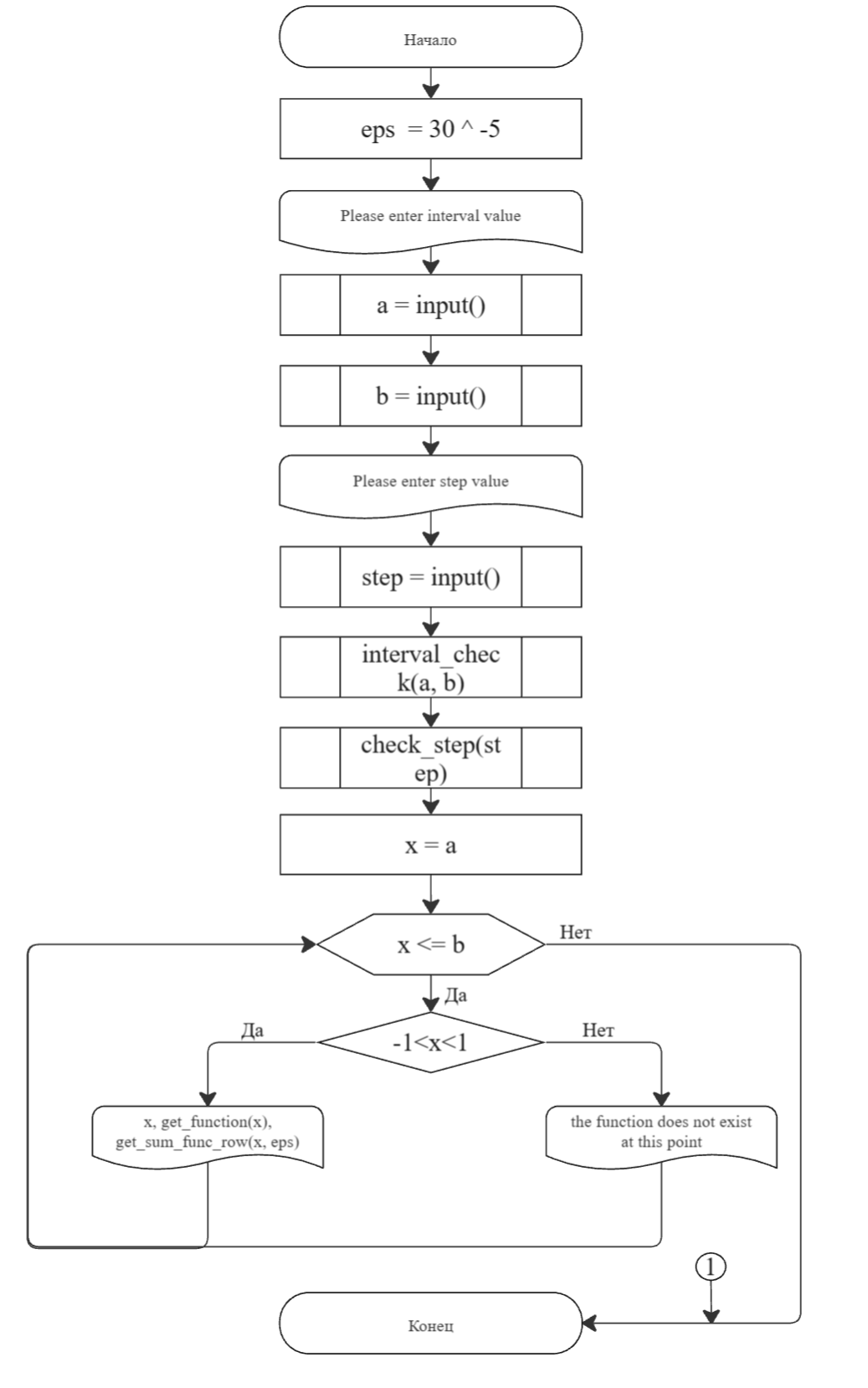


Рисунок  – Блок-схема основного алгоритма

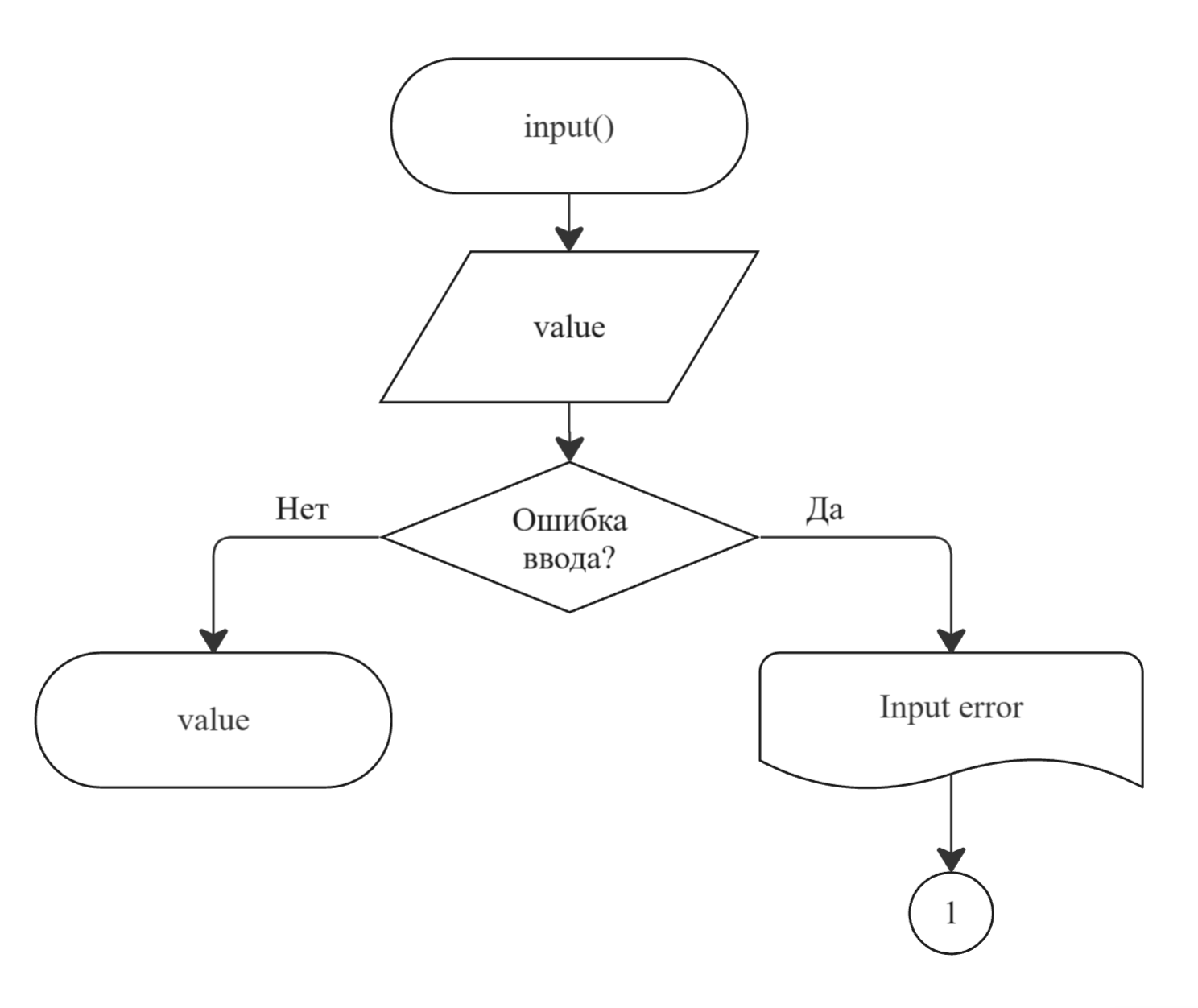


Рисунок – Блок-схема функции input()

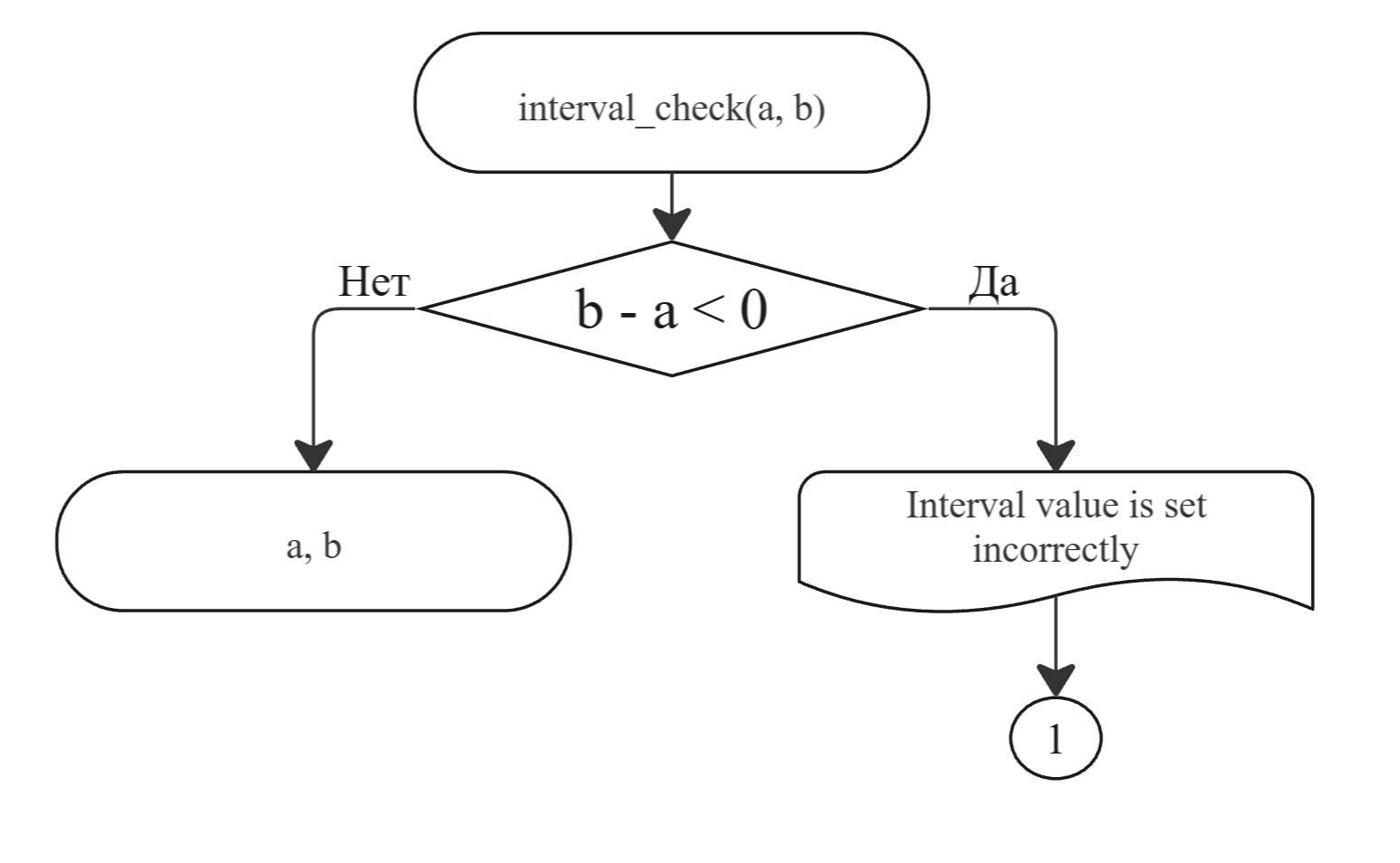


Рисунок 33 – Блок-схема функции interval\_check(a, b)

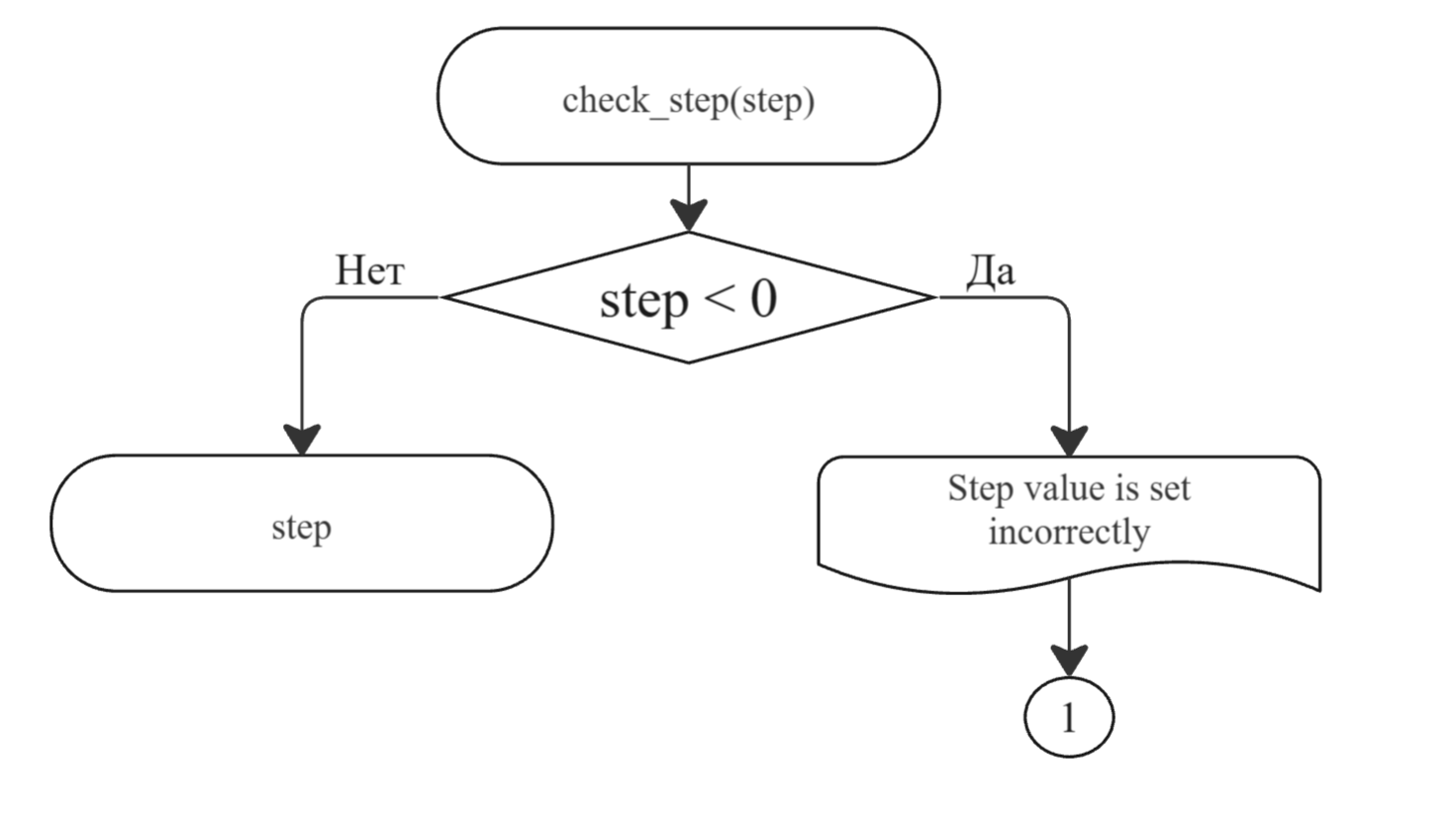


Рисунок  – Блок-схема функции check\_step(step)

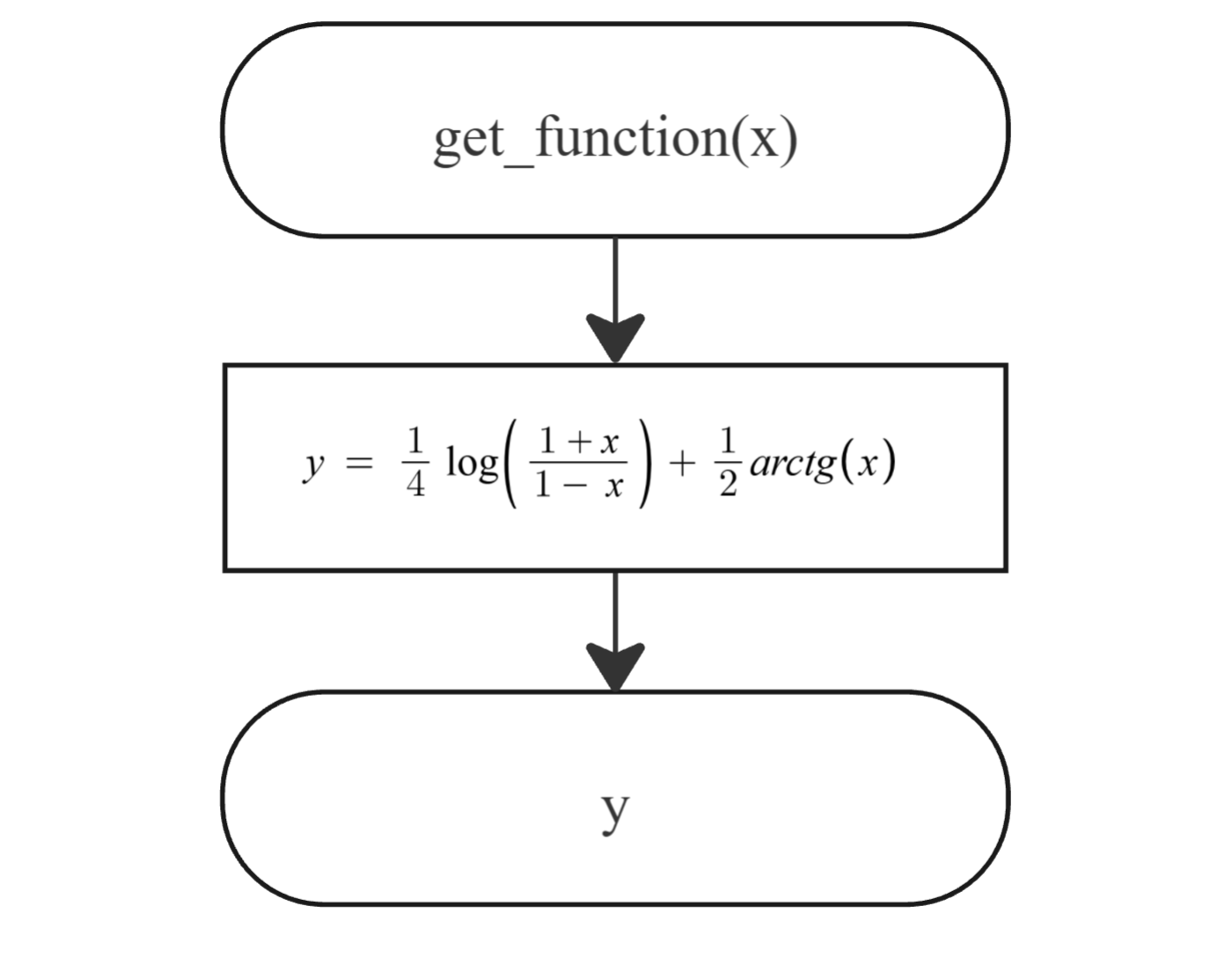


Рисунок  – Блок-схема функции get\_function(x)

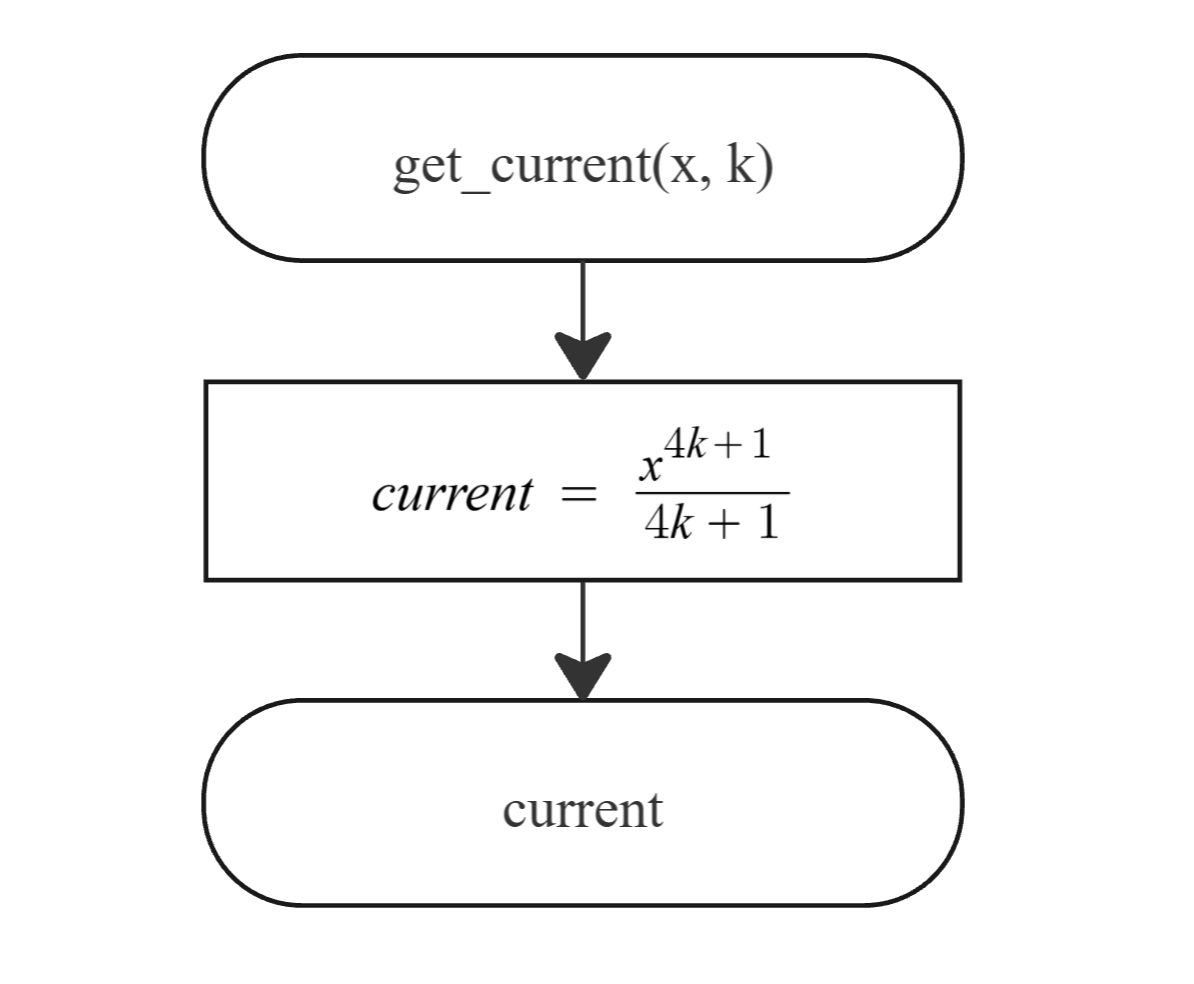


Рисунок  – Блок-схема функции get\_current(x, k)

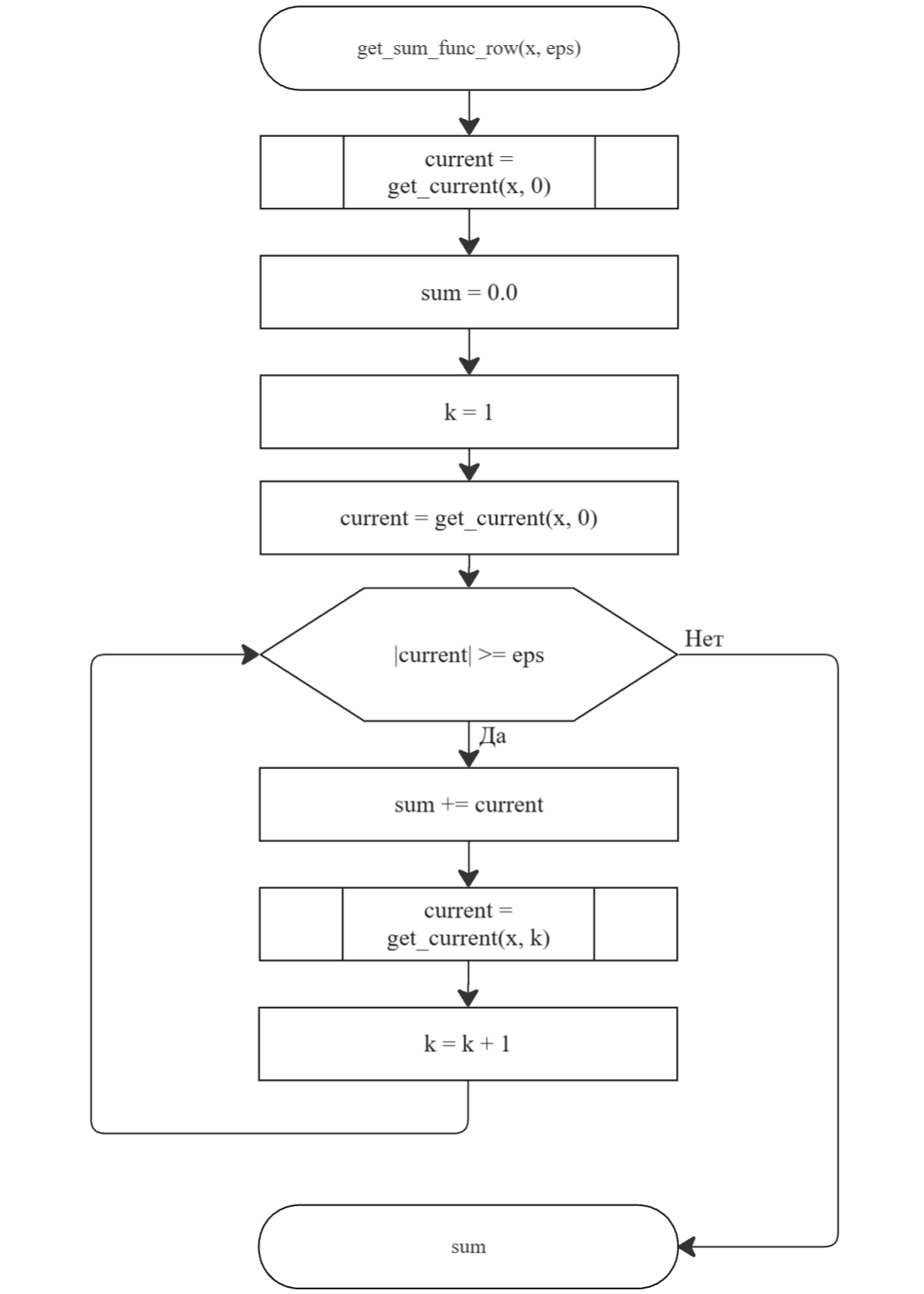


Рисунок  – Блок-схема функции get\_sum\_func\_row(x, eps)

3.3 Код программы на языке С

#include <stdio.h>

#include <math.h>

#include <errno.h>

#include <stdlib.h>

#include <float.h>

/\*\*

\* @brief получает на вход число из потока ввода

\* @return возвращает полученное число, иначе вернет ошибку

\*/

double input(void);

/\*\*

\* @brief функция, проверяющая заданный интервал

\* @param a начальная точка заданного интервала

\* @param b конечная точка заданного интервала

\* @return возвращает ошибку в случае, если интервал был задан неверно

\*/

void interval\_check(const double a, const double b);

/\*\*

\* @brief функция, проверяющая заданный шаг

\* @param step значение заданного шага

\* @return возвращает ошибку в случае, если шаг был задан неверно

\*/

void check\_step(const double step);

/\*\*

\* @brief рассчитывает значение функции в заданной точке

\* @param x значение аргумента

\* @return рассчитанное значение функции

\*/

double get\_function(const double x);

/\*\*

\* @brief рассчитывает рекуррентное выражение

\* @param x значение аргумента

\* @param k значение индекса элемента

\* @return рассчитанное рекуррентное значение

\*/

double get\_current(const double x, const int k);

/\*\*

\* @brief рассчитывает сумму функционального ряда

\* @param a начальная точка интервала

\* @param b конечная точка интервала

\* @param eps заданная точность

\* @return рассчитанное значение суммы функционального ряда

\*/

double get\_sum\_func\_row(const double x, const double eps);

/\*\*

\* @brief точка входа в программу

\* @return 0 в случае успеха

\*/

int main(void)

{

const double eps = pow(30, -5);

printf("Please enter interval value\n");

const double a = input();

const double b = input();

printf("Please enter step value\n");

const double step = input();

interval\_check(a, b);

check\_step(step);

for (double x = a; x <= b + DBL\_EPSILON; x += step)

{

if (x > -1 - DBL\_EPSILON && x < 1 + DBL\_EPSILON) { //изменение ООФ//

printf("the function does not exist at this point\n");

}else {

printf("\nx = %.2f\t\tf(x) = %.6f\t\tS(x) = %.6f\n", x, get\_function(x), get\_sum\_func\_row(x, eps));

}

}

return 0;

}

double input(void)

{

double value = 0.0;

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

if (result != 1)

{

errno = EIO;

perror("Input error\n");

exit(EXIT\_FAILURE);

}

return value;

}

void interval\_check(const double a, const double b)

{

if (b - a < DBL\_EPSILON)

{

errno = EINVAL;

perror("Interval value is set incorrectly\n");

exit(EXIT\_FAILURE);

}

}

void check\_step(const double step)

{

if (step <= DBL\_EPSILON)

{

errno = EINVAL;

perror("Step value is set incorrectly\n");

exit(EXIT\_FAILURE);

}

}

double get\_function(const double x)

{

return (1.0 / 4.0) \* log((1 + x) / (1 - x)) + (1.0 / 2.0) \* atan(x);

}

double get\_current(const double x, int k)

{

return pow(x, 4 \* k + 1) / (4 \* k + 1);

}

double get\_sum\_func\_row(const double x, const double eps)

{

double current = get\_current(x, 0); // Начальное значение current

double sum = 0.0;

int k = 1;

while (fabs(current) >= eps)

{

sum += current;

current = get\_current(x, k);

k++;

}

return sum;

}

3.4 Результат выполнения программы

Ниже представлены результаты выполнения программы (Рисунок 38, Рисунок 39, Рисунок 40, Рисунок 41, Рисунок 42)

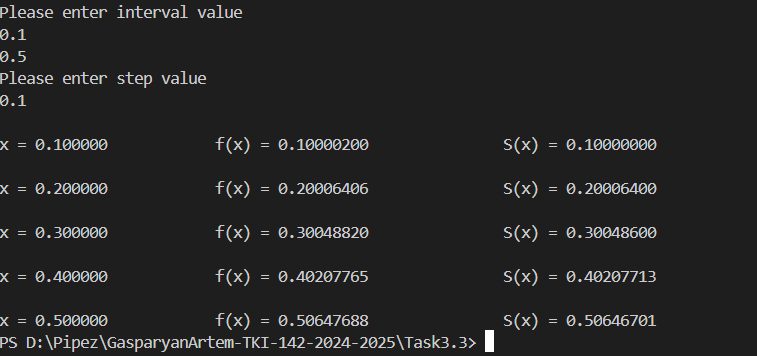


Рисунок  – Первый результат выполнения программы

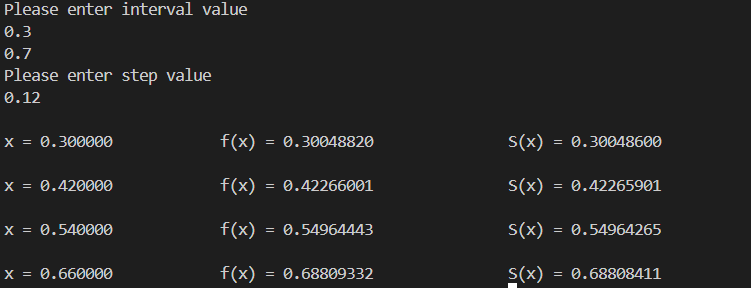


Рисунок  – Второй результат выполнения программы

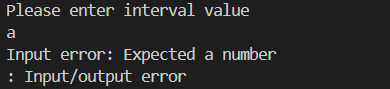


Рисунок  – Проверка функции input()

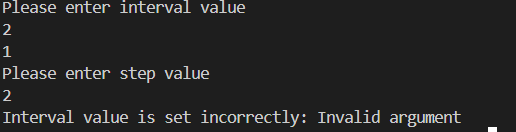


Рисунок  – Проверка функции interval\_check(a, b)

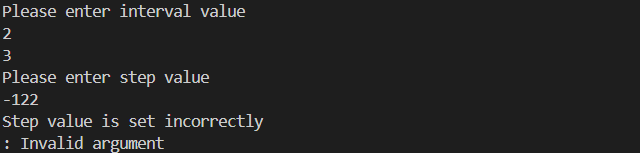


Рисунок  – Проверка функции check\_step(step)

3.5 Выполнение тестовых примеров

В программе MS Excel были выполнены тестовые примеры. Результат их выполнения представлен ниже (Рисунок 43, Рисунок 44).

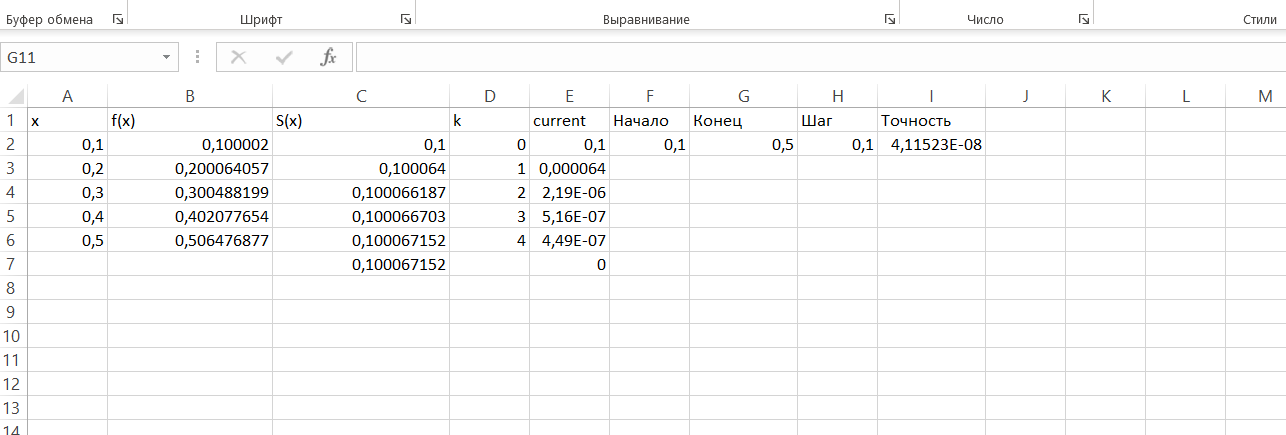


Рисунок  – Выполнение первого тестового примера

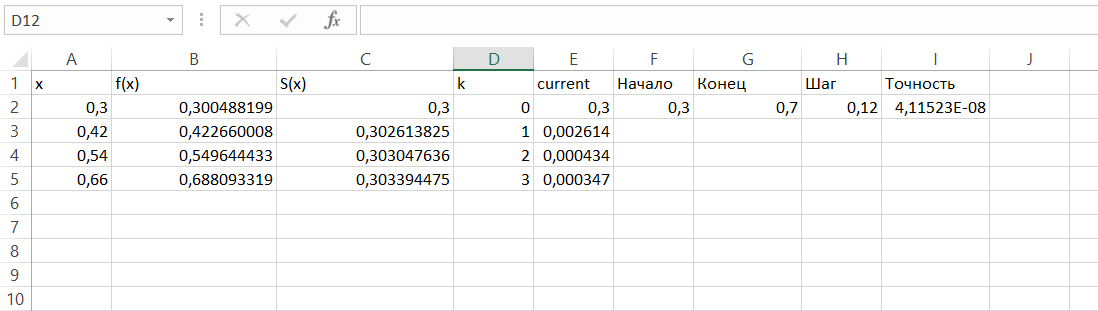


Рисунок  – Выполнение второго тестового примера

3.6 Отметка о выполнение задания

Отметка о выполнение задания в GitHub (Рисунок 45).

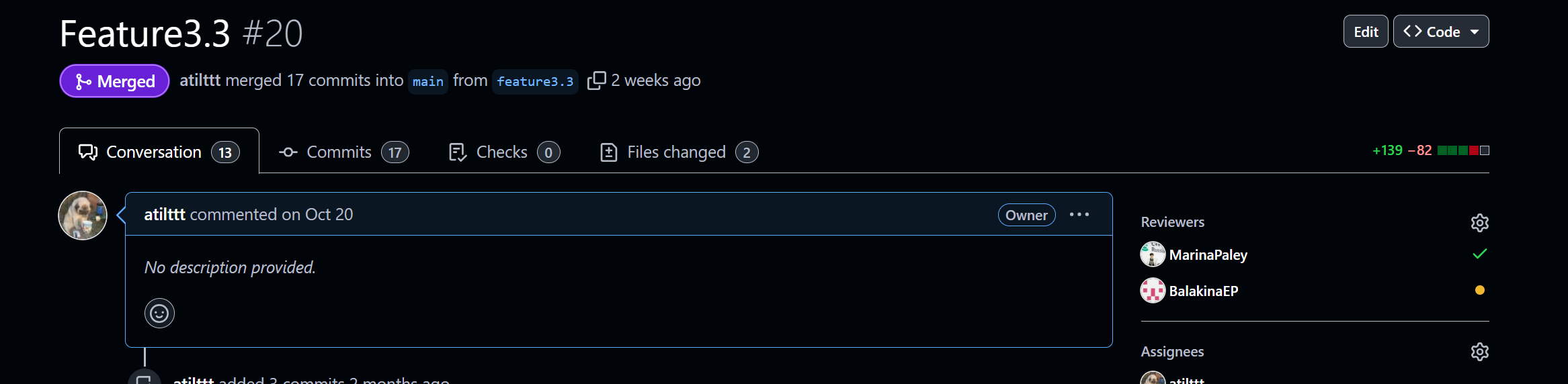


Рисунок  – Отметка о выполнение задания