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 1

Выполнил: ст. гр. ТКИ - 142

Шабин Константин Олегович

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

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

Москва 2023

Оглавление

[1. Задание 3.1 3](#_Toc156766590)

[1.1. Формулировка задания 3](#_Toc156766591)

[1.2. Блок-схема алгоритма 4](#_Toc156766592)

[1.3. Код задания 3.1 8](#_Toc156766593)

[1.4. Решение тестового примера 10](#_Toc156766594)

[1.5. Расчёт тестового примера в Excel 11](#_Toc156766595)

[1.6. Зачёт задания в GitHub 11](#_Toc156766596)

[2. Задание 3.2 12](#_Toc156766597)

[2.1. Формулировка задания 12](#_Toc156766598)

[2.2. Блок-схема алгоритма 13](#_Toc156766599)

[2.3. Код задания 3.2 17](#_Toc156766600)

[2.4. Решение тестового примера 19](#_Toc156766601)

[2.5. Решение тестовых примеров в Excel 20](#_Toc156766602)

[2.6. Зачёт задания в GitHub 21](#_Toc156766603)

# 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 |
| 1 |  |  |

* 1. Блок-схема алгоритма

Блок-схема алгоритма представлена на рисунках (Рисунок 1, Рисунок 2, Рисунок 3, Рисунок 4, Рисунок 5, Рисунок 6)

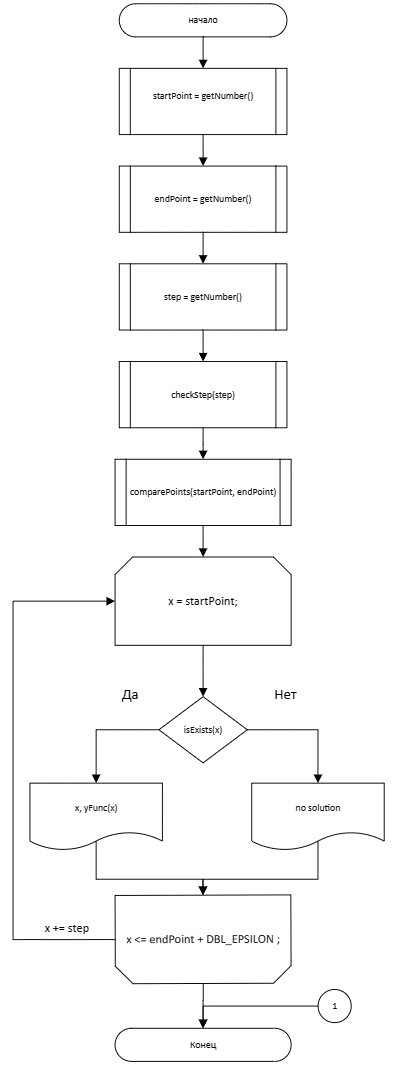


Рисунок 1 – Блок-схема функции main()

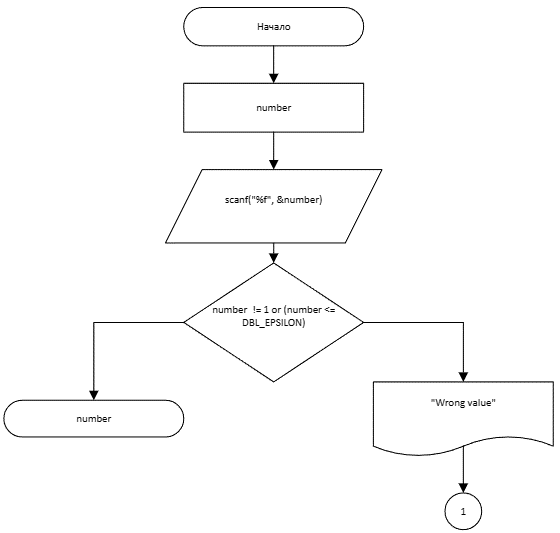


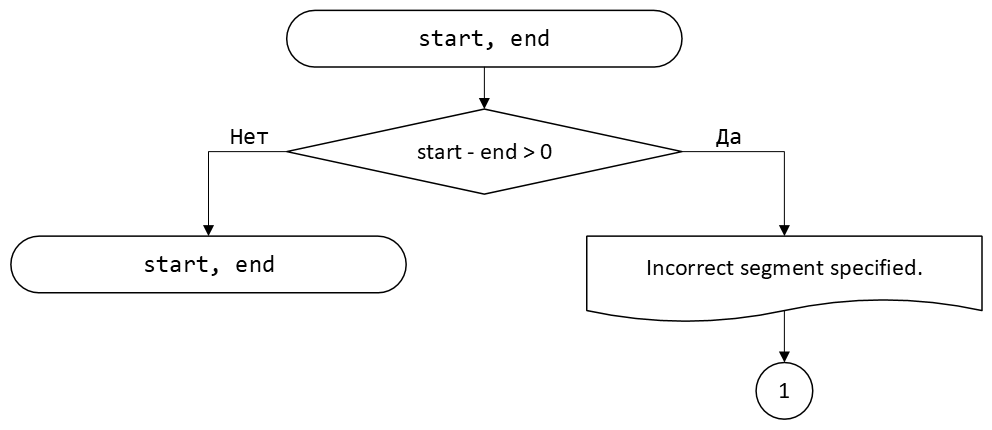
Рисунок 2 – Блок-схема функции getNumber()

Рисунок 3 – Блок-схема к функции comparePoints(startPoint, endPoint)

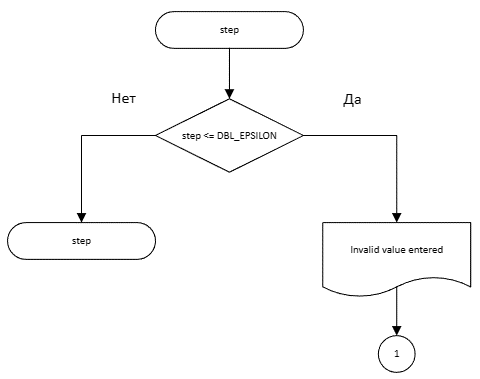


Рисунок 4 – Блок-схема функции checkStep(step)

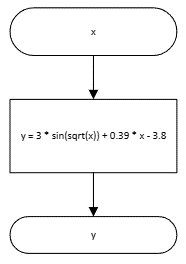


Рисунок 5 – Блок-схема функции yFunc(x)

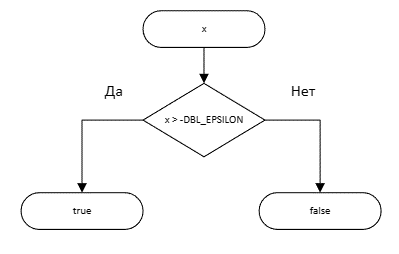


Рисунок 6 – Блок-схема функции isExists(x)

* 1. Код задания 3.1

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <stdbool.h>

#include <float.h>

/\*\*

\* @brief расситывает значение у

\* @param x значение х получаемое из цикла for

\* @return возвращает значение у

\*/

double yFunc(double x);

/\*\*

\* @brief проверяет, существует ли функция в точке х

\* @param x аргумент функции

\* @return возвращает true если существует

\*/

bool isExists(const double x);

/\*\*

\* @brief проверка введеного значения

\* @return возвращает значение если ввод правильный, иначе выводит сообщение об ошибке

\*/

double getNumber();

/\*\*

\* @brief проверяет правильность ввода шага

\* @param step введенное значение шага расчета

\* @return возвращает true если ввод правильный

\*/

bool checkStep(double step);

/\*\*

\* @brief проверяет правильность ввода начальной и конечной точек расчета

\* @param startPoint введенное значение начальной точки расчета

\* @param endPoint введенное значение конечной точки расчета

\* @return возвращает true если ввод правильный

\*/

bool comparePoints(double startPoint, double endPoint);

/\*\*

\* @brief Точка входа в программу

\* @return Возвращает 0, если программа работает корректно, иначе 1

\*/

int main()

{

double startPoint = getNumber();

double endPoint = getNumber();

double step = getNumber();

checkStep(step);

comparePoints(startPoint, endPoint);

for(double x = startPoint; x <= endPoint + DBL\_EPSILON ; x += step)

{

if (isExists(x))

{

printf("\n %7.4lf %7.4lf", x, yFunc(x));

}

else

{

printf("\n %7.4lf %s", x, "no solution");

}

}

return 0;

}

double yFunc(double x)

{

return 3 \* sin(sqrt(x)) + 0.39 \* x - 3.8;

}

double getNumber()

{

double number;

if (scanf("%lf", &number) != 1)

{

errno = EIO;

perror("Invalid value entered");

abort();

}

return number;

}

bool isExists(const double x)

{

if (x > -DBL\_EPSILON)

{

return true;

}

return false;

}

bool checkStep(double step)

{

if (step <= DBL\_EPSILON)

{

errno = EIO;

perror("Invalid value entered");

abort();

}

return true;

}

bool comparePoints(double startPoint, double endPoint)

{

if (startPoint - endPoint >= DBL\_EPSILON)

{

errno = EIO;

perror("Invalid value entered");

abort();

}

return true;

}

* 1. Решение тестового примера

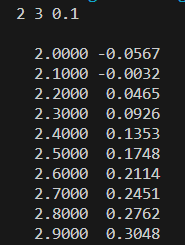


Рисунок 7 – Решение тестового примера



Рисунок 8 – Вывод программы, когда начало сегмента больше конца



Рисунок 9 – Вывод программы, когда начало сегмента – буква



Рисунок 10 – Вывод программы, когда конец отрезка – буква



Рисунок 11 – Вывод программы, когда шаг функции отрицательный



Рисунок 12 – Вывод программы, когда шаг функции – буква

* 1. Расчёт тестового примера в Excel

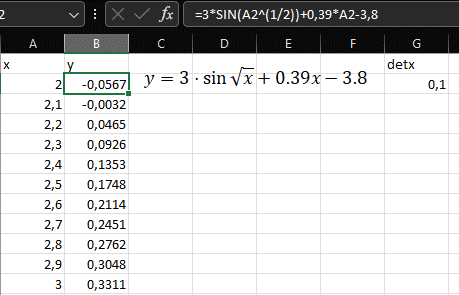


Рисунок 13 – Решение тестового примера в Excel

* 1. Зачёт задания в GitHub

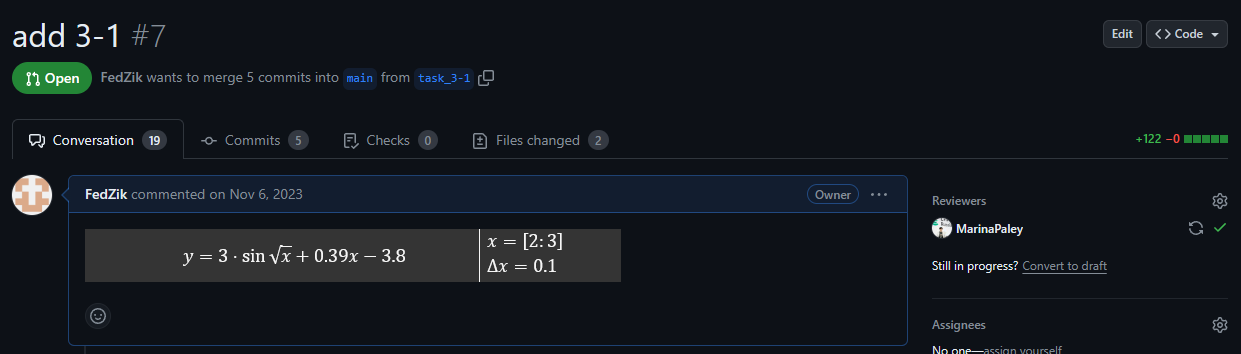


Рисунок 14 – Зачёт задания в GitHub

# 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** |
| 1 |  |

* 1. Блок-схема алгоритма

Блок-схема алгоритма представлена на рисунках (Рисунок 15, Рисунок 16, Рисунок 17, Рисунок 18, Рисунок 19, Рисунок 20)

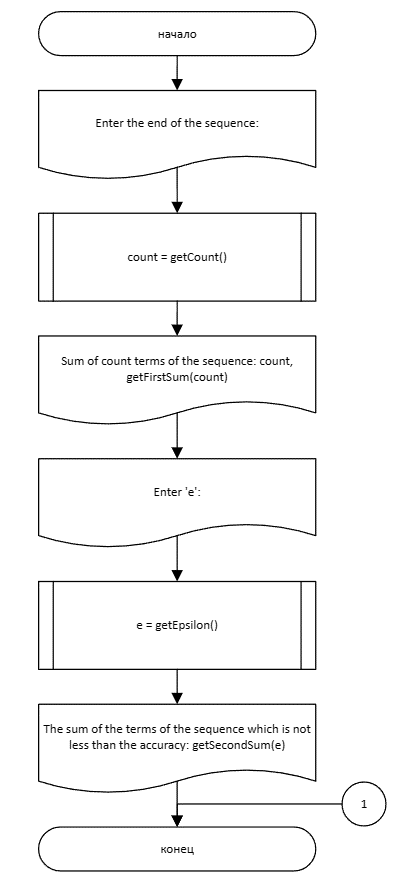


Рисунок 15 – Блок-схема функции main()

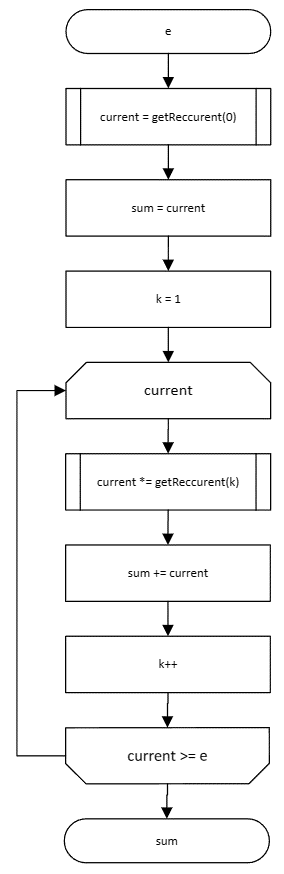


Рисунок 16 – Блок-схема функции getSecondSum(e)

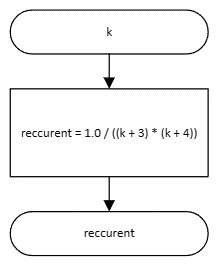


Рисунок 17 – Блок-схема функции getReccurent(k)

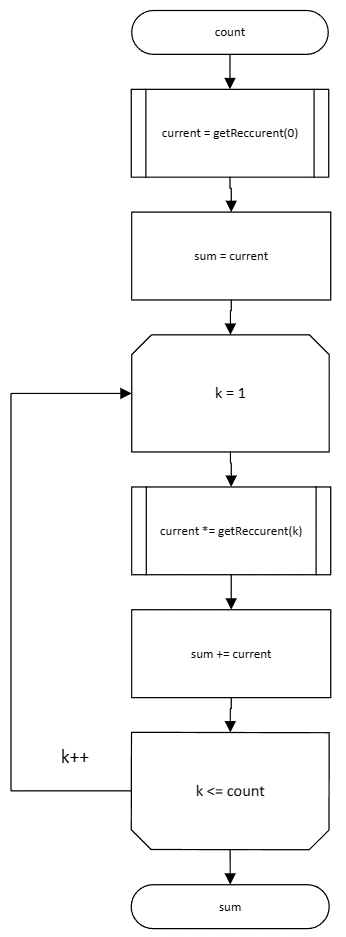


Рисунок 18 – Блок-схема функции getFirstSum(count)

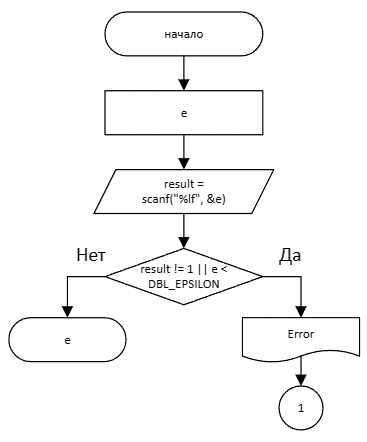


Рисунок 19 – Блок-схема функции get\_epsilon()

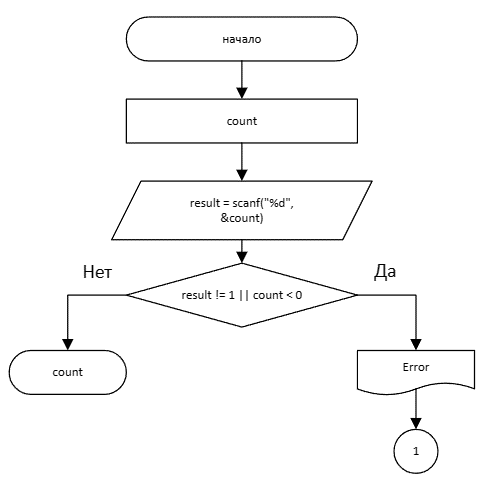


Рисунок 20 – Блок-схема функции getCount()

* 1. Код задания 3.2

#include <stdio.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 getCount();

/\*\*

\* @brief Функция расчитывающая сумму членов последовательности.

\* @param count - количество членов последовательности.

\* @return Сумма последовательности.

\*/

double getFirstSum(int count);

/\*\*

\* @brief Функция расчитывающая рекурентный член последовательности.

\* @param k номер члена последовательности.

\* @return Значение рекурентного члена последовательности.

\*/

double getReccurent(int k);

/\*\*

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

\* @return Точность вычислений.

\*/

double getEpsilon();

/\*\*

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

\* @param e - точность вычислений.

\* @return Сумма.

\*/

double getSecondSum(const double e);

/\*\*

\* @brief Точка входа в программу.

\* @return Возвращает 0 в случае успеха.

\*/

int main()

{

printf("%s", "Enter the end of the sequence: ");

int count = getCount();

printf("Sum of %d terms of the sequence: %.15lf \n", count, getFirstSum(count));

printf("Enter 'e': ");

const double e = getEpsilon();

printf("The sum of the terms of the sequence which is not less than the accuracy: %.15lf\n", getSecondSum(e));

return 0;

}

int getCount()

{

int count;

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

if (result != 1 || count < 0)

{

errno = EIO;

perror("Error");

abort();

}

return count;

}

double getFirstSum(int count)

{

double current = getReccurent(0);

double sum = current;

for (int k = 1; k <= count; k++)

{

current \*= getReccurent(k);

sum += current;

}

return sum;

}

double getReccurent(int k)

{

return 1.0 / ((k + 3) \* (k + 4));

}

double getEpsilon()

{

double e;

double result = scanf("%lf", &e);

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

{

errno = EIO;

perror("Error");

abort();

}

return e;

}

double getSecondSum(const double e)

{

double current = getReccurent(0);

double sum = current;

int k = 1;

while (current >= e)

{

current \*= getReccurent(k);

sum += current;

k++;

}

return sum;

}

* 1. Решение тестового примера

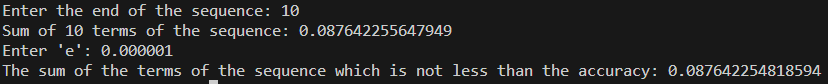


Рисунок 21 – Решение тестового примера



Рисунок 22 – Вывод программы, когда количество членов последовательности неположительное



Рисунок 23 – Вывод программы, когда количество членов последовательности – буква

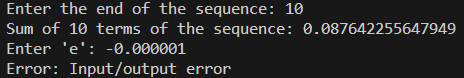


Рисунок 24 – Вывод программы, когда точность вычислений отрицательная

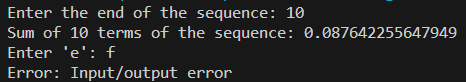


Рисунок 25 – Вывод программы, когда точность вычислений – буква

* 1. Решение тестовых примеров в Excel

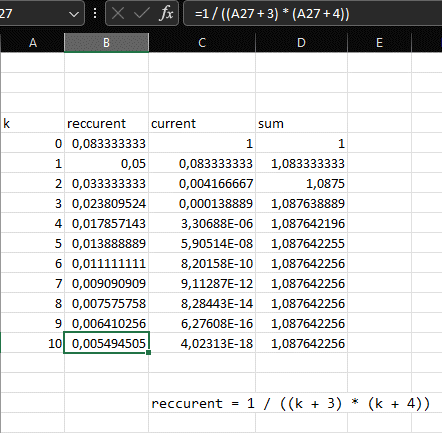


Рисунок 26 – Расчёт суммы первых 10 членов последовательности в Excel

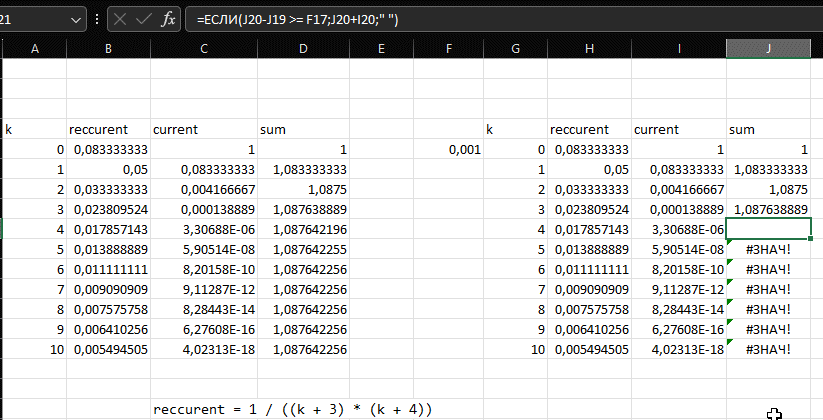


Рисунок 27 – Расчёт суммы членов последовательности, не меньших заданного числа e

* 1. Зачёт задания в GitHub

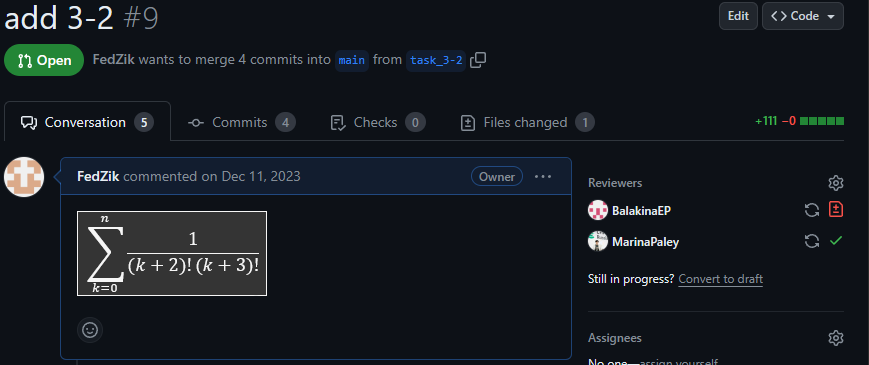


Рисунок 28 – Зачёт задания в GitHub