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

Горковец Анна Сергеевна

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

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

Москва 2023

Оглавление

[1. Задание 3.1 3](#_Toc155785062)

[1.1. Формулировка задания 3](#_Toc155785063)

[1.2. Блок-схема алгоритма 4](#_Toc155785064)

[1.3. Код задания 3.1 8](#_Toc155785065)

[1.4. Решение тестового примера 9](#_Toc155785066)

[1.5. Расчёт тестового примера в Excel 10](#_Toc155785067)

[1.6. Зачёт задания в GitHub 11](#_Toc155785068)

[2. Задание 3.2 12](#_Toc155785069)

[2.1. Формулировка задания 12](#_Toc155785070)

[2.2. Блок-схема алгоритма 13](#_Toc155785071)

[2.3. Решение тестового примера 19](#_Toc155785072)

[2.4. Решение тестовых примеров в Excel 21](#_Toc155785073)

[2.5. Зачёт задания в GitHub 21](#_Toc155785074)

[3. Задание 3.3 22](#_Toc155785075)

[3.1. Формулировка задания 22](#_Toc155785076)

[3.2. Блок-схема алгоритма 23](#_Toc155785077)

[3.3. Код задания 3.3 27](#_Toc155785078)

[3.4. Решение тестового примера 31](#_Toc155785079)

[3.5. Зачёт задания в GitHub 32](#_Toc155785080)

# 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 |
| 5 |  |  |

* 1. Блок-схема алгоритма

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

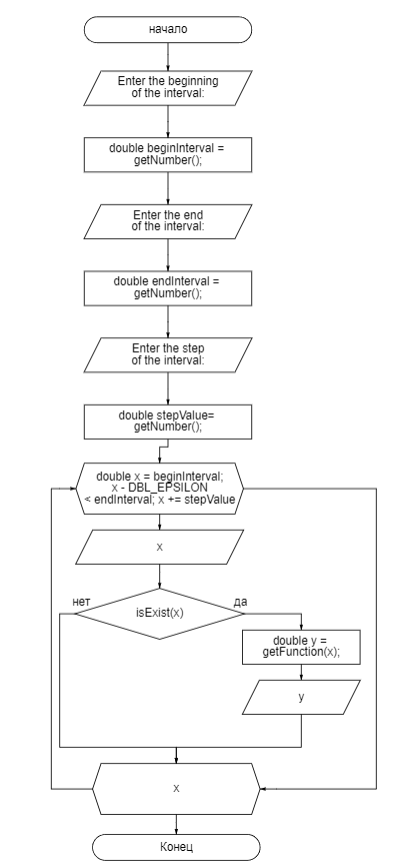


Рисунок 1 – Блок-схема функции main()

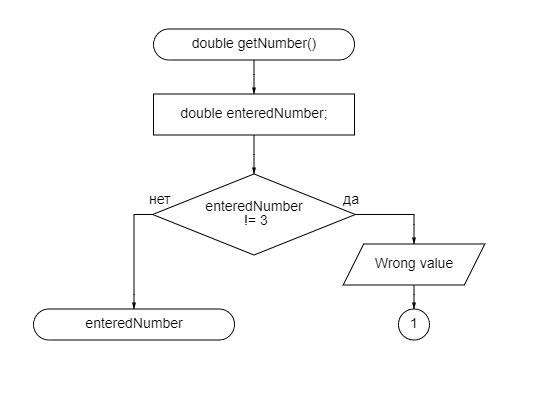


Рисунок 2 – Блок-схема функции getNumber ()

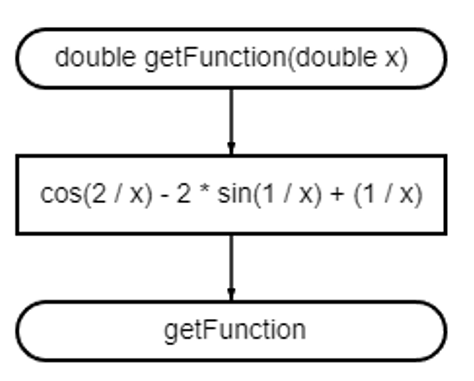


Рисунок 3 – Блок-схема к функции getFunction()

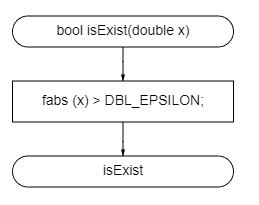


Рисунок 4 – Блок-схема функции isExist()

* 1. Код задания 3.1

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <float.h>

#include <stdbool.h>

#include <errno.h>

#define printf\_s printf

#define scanf\_s scanf

/\*\*

\* @brief Рассчитывает значение функции (табулирование функции)

\* @param x - значение переменной x

\* @return Возвращает значение функции

\*/

double getFunction(double x);

/\*\*

\* @brief Считывает и проверяет ввод переменной

\* @return Возвращает считанное значение

\*/

double getNumber();

/\*\*

\* @brief Проверка принадлежности x к ООФ

\* @param x - значение переменной x

\* @return Возвращает значение функции

\*/

bool isExist(double x);

/\*\*

\* @brief Точка входа в программу

\* @return Возвращает 0, если программа работает верно

\*/

int main()

{

printf\_s("Enter the beginning of the interval: ");

double beginInterval = getNumber();

printf\_s("Enter the end of the interval: ");

double endInterval = getNumber();

printf\_s("Enter step: ");

double stepValue = getNumber();

for (double x = beginInterval; x - DBL\_EPSILON < endInterval; x += stepValue)

{

printf\_s("x = %lf", x);

if (isExist(x))

{

double y = getFunction(x);

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

}

else

printf\_s("y не существует\n");

}

return 0;

}

double getNumber()

{

double enteredNumber;

if (scanf\_s("%lf", &enteredNumber) != 1)

{

errno = EIO;

perror("Wrong value");

exit(1);

}

return enteredNumber;

}

double getFunction(double x)

{

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

}

bool isExist (double x)

{

return fabs (x) > DBL\_EPSILON;

}

* 1. Решение тестового примера

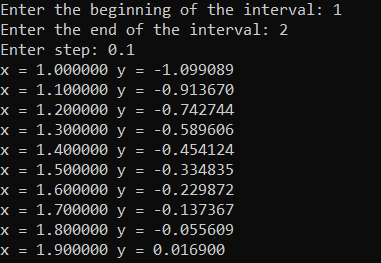


Рисунок 5 – Решение тестового примера



Рисунок 6 – Вывод программы, когда начало интервала – буква

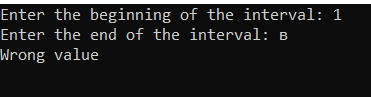


Рисунок 7 – Вывод программы, когда конец интервала – буква

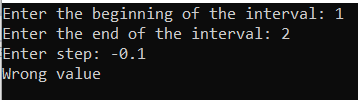


Рисунок 8 – Вывод программы, когда шаг функции отрицательный

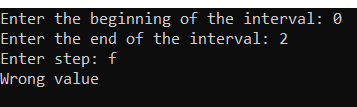


Рисунок 9 – Вывод программы, когда шаг функции – буква

* 1. Расчёт тестового примера в Excel

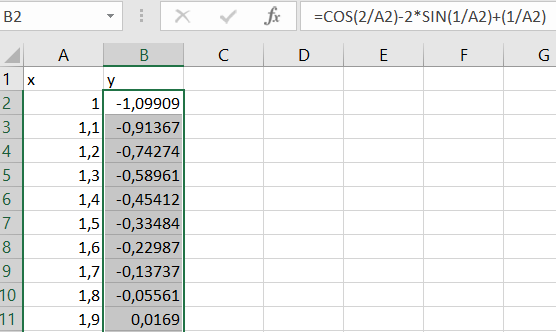


Рисунок 10 – Решение тестового примера в Excel

* 1. Зачёт задания в GitHub

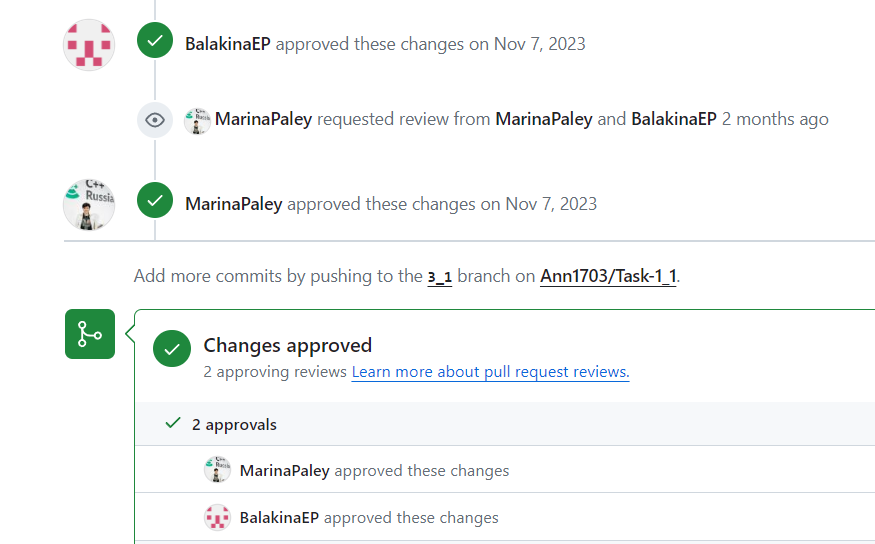


Рисунок 11 – Зачёт задания в GitHub

# 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** |
| 5 |  |

* 1. Блок-схема алгоритма

Блок-схема алгоритма представлена на рисунках (Рисунок 13, Рисунок 14, Рисунок 15, Рисунок 16, Рисунок 17)

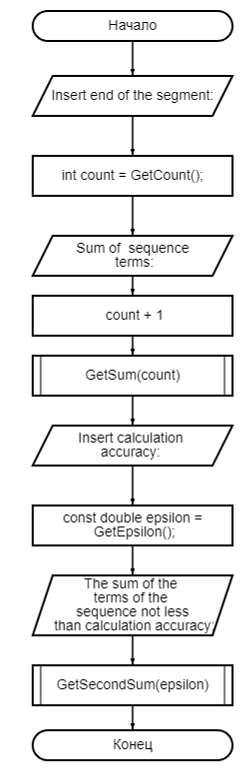


Рисунок 12 – Блок-схема функции main()

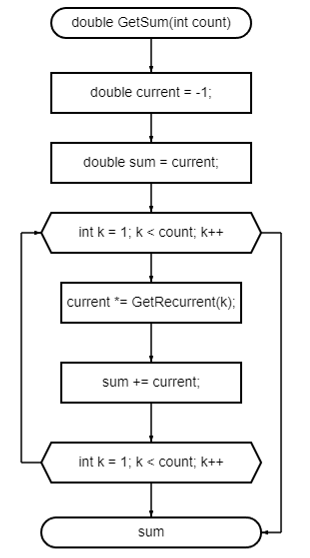


Рисунок 13 – Блок-схема функции GetSum()

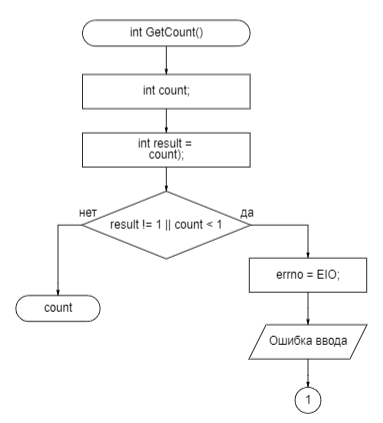


Рисунок 14 – Блок-схема функции GetCount()

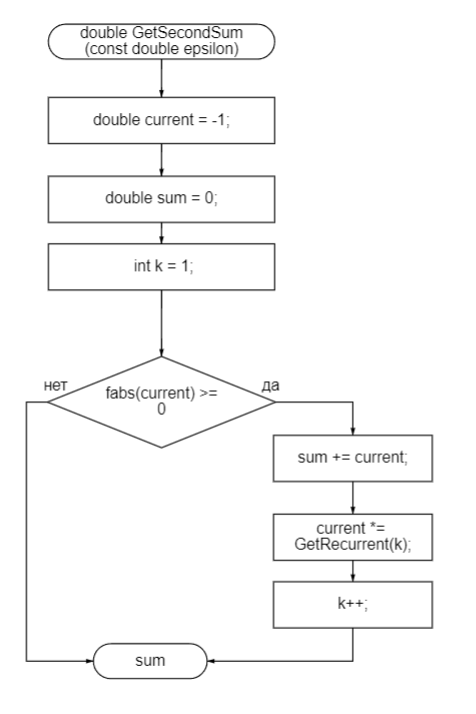


Рисунок 15 – Блок-схема функции double GetSecondSum()

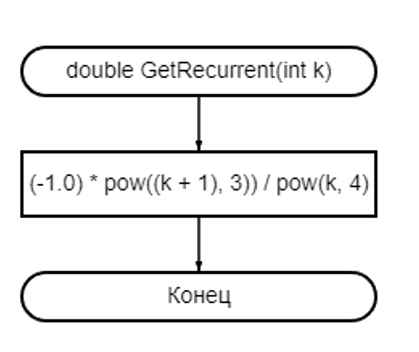


Рисунок 16 – Блок-схема функции GetRecurrent(int k)

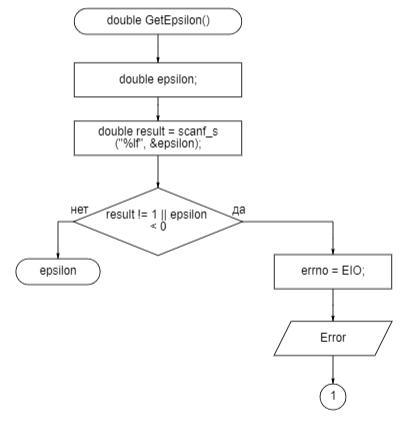


Рисунок 17 – Блок-схема функции GetEpsilon()

Код задания 3.2

#include <stdio.h>

#include <math.h>

#include <float.h>

#include <errno.h>

#include <stdlib.h>

#define scanf\_s scanf

#define printf\_s printf

/\*\*

\* @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 GetSum(int count);

/\*\*

\* @brief Функция расчитывающая рекурентный член последовательности.

\* @param k - номер члена последовательности.

\* @return Значение рекурентного члена последовательности.

\*/

double GetRecurrent(int k);

/\*\*

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

\* @return Точность вычислений.

\*/

double GetEpsilon();

/\*\*

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

\* @param epsilon - точность вычислений.

\* @return Сумма.

\*/

double GetSecondSum(const double epsilon);

/\*\*

\* @brief Точка входа в программу.

\* @return Возвращает 0 в случае успеха.

\*/

/\*\*

\* @brief Точка входа в программу.

\* @return Возвращает 0 в случае успеха.

\*/

int main()

{

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

int count = GetCount();

printf\_s("Sum of %d sequence terms: %lf \n", count, GetSum(count));

printf\_s("Insert calculation accuracy: ");

const double epsilon = GetEpsilon();

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

return 0;

}

int GetCount()

{

int count;

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

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

{

errno = EIO;

perror("Ошибка ввода");

abort();

}

return count;

}

double GetSum(int count)

{

double current = -1;

double sum = current;

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

{

current \*= GetRecurrent(k);

sum += current;

}

return sum;

}

double GetRecurrent(int k)

{

return ((-1.0) \* pow((k + 1), 3)) / pow(k, 4);

}

double GetEpsilon()

{

double epsilon;

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

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

{

errno = EIO;

perror("Error");

abort();

}

return epsilon;

}

double GetSecondSum(const double epsilon)

{

double current = -1;

double sum = current;

int k = 1;

while (fabs(current) >= epsilon)

{

current \*= GetRecurrent(k);

sum += current;

k++;

}

return sum;

}

* 1. Решение тестового примера

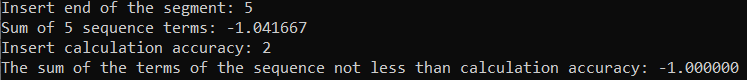


Рисунок 18 – Решение тестового примера

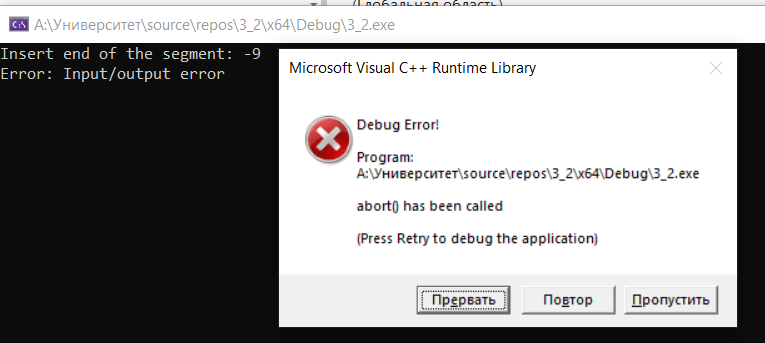


Рисунок 19 – Вывод программы, когда количество членов последовательности неположительное

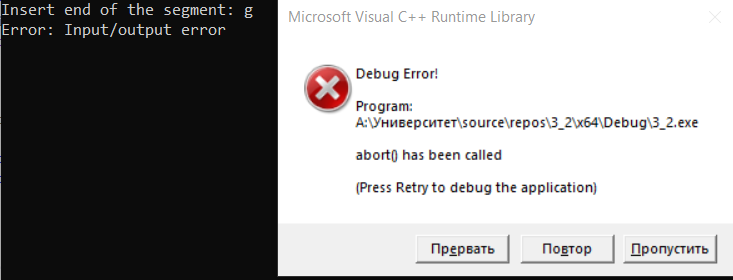


Рисунок 20 – Вывод программы, когда количество членов последовательности – буква

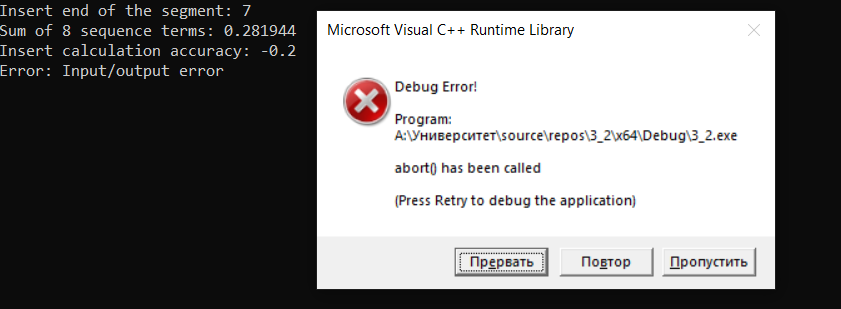


Рисунок 21 – Вывод программы, когда точность вычислений отрицательная

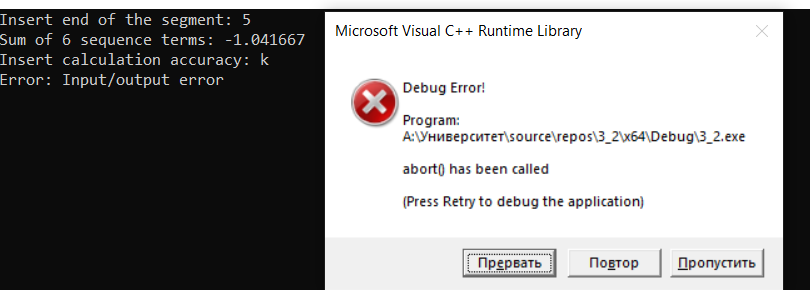


Рисунок 22 – Вывод программы, когда точность вычислений – буква

* 1. Решение тестовых примеров в Excel

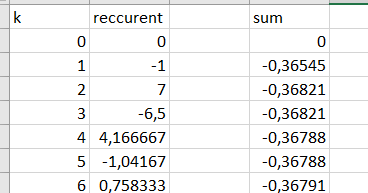


Рисунок 23 – Расчёт суммы первых 6 членов последовательности в Excel

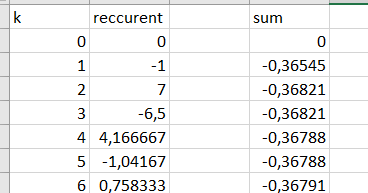


Рисунок 24 – Расчёт суммы членов последовательности, не меньших заданного числа e

* 1. Зачёт задания в GitHub

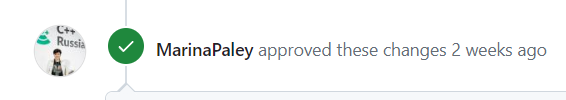


Рисунок 25 – Зачёт задания в GitHub

# 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 |
| 5 |  |  |  | 30-5 |

* 1. Блок-схема алгоритма

Блок-схема алгоритма представлена на рисунках (Рисунок 26, Рисунок 27, Рисунок 28, Рисунок 29, Рисунок 30, Рисунок 31, Рисунок 32, Рисунок 33)

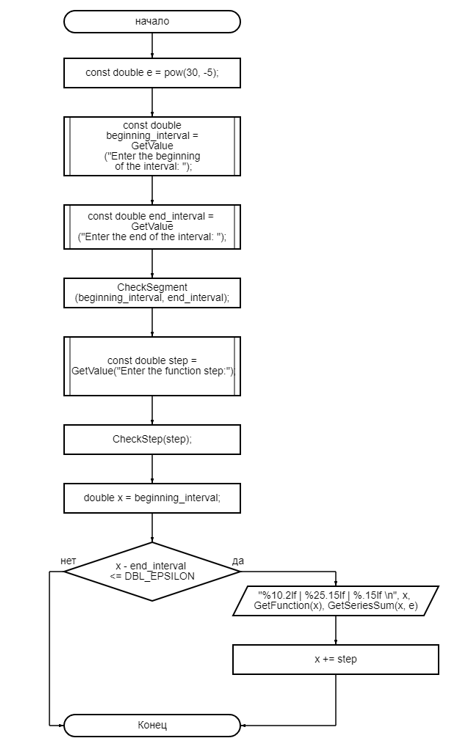


Рисунок 26 – Блок-схема функции main()

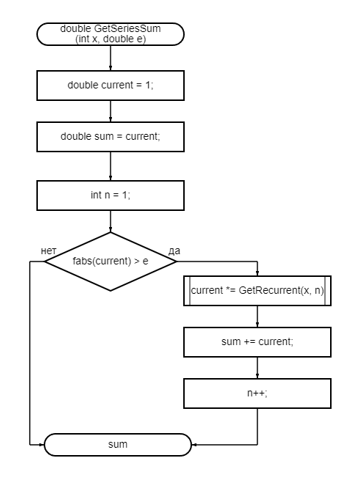


Рисунок 27 – Блок-схема функции GetSeriesSum()

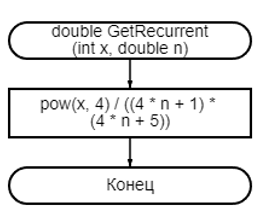


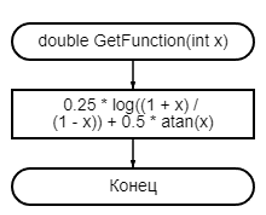
Рисунок 28 – Блок-схема функции GetRecurrent()

Рисунок 29 – Блок-схема функции GetFunction()

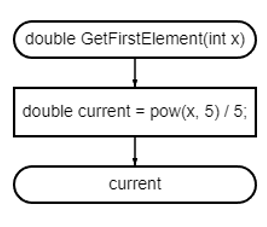


Рисунок 30 – Блок-схема функции GetFirstElement()

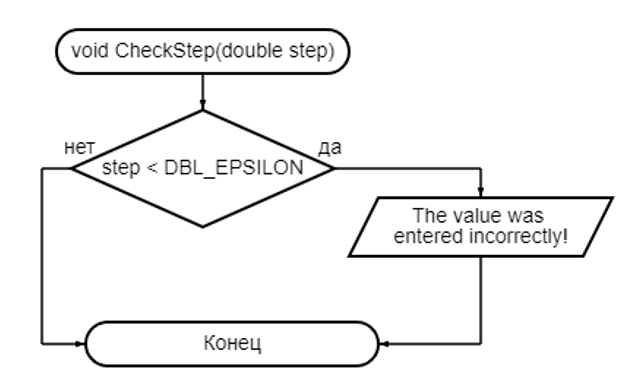


Рисунок 31 – Блок-схема функции CheckStep()

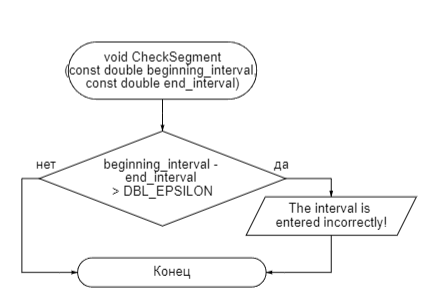


Рисунок 32 – Блок-схема функции CheckSegment()

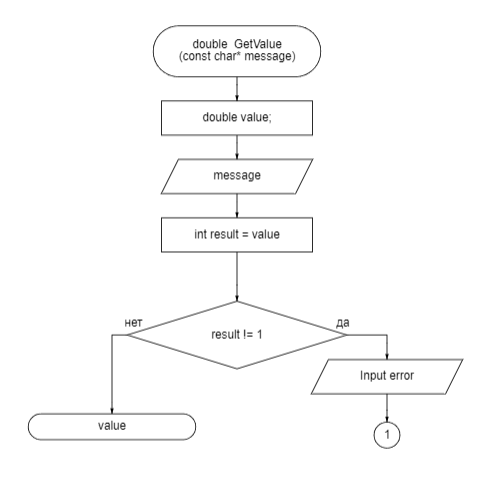


Рисунок 33 – Блок-схема функции GetValue()

* 1. Код задания 3.3

﻿#include <locale.h>

#include <errno.h>

#include <math.h>

#include <stdlib.h>

#include <float.h>

#include <stdio.h>

/\*\*

\* @brief Функция для вычисления рекуррентного члена ряда.

\* @param x Параметр функции, аппроксимируемой рядом.

\* @param n Индекс члена ряда.

\* @return Значение рекуррентного члена.

\*/

double GetRecurrent(int x, double n);

/\*\*

\* @brief Функция для вычисления суммы ряда.

\* @param x Показатель степени.

\* @return Результат 3-го значения в степени x.

\*/

double GetSeriesSum(int x, double e);

/\*\*

\* @brief Функция для вычисления значения функции.

\* @param x - аргумент функции.

\* @return Значение функции.

\*/

double GetFunction(int x);

/\*\*

\* @brief Функция проверки ввода шага на правильность.

\* @param step - шаг функции.

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

\*/

void CheckStep(double step);

/\*\*

\* @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 GetValue(const char\* message);

/\*\*

\* @brief Функция проверки отрезка на существование.

\* @param beginning\_interval - начало отрезка.

\* @param end\_interval - конец

отрезка.

\*/

void CheckSegment(const double beginning\_interval, const double end\_interval);

/\*\*

\* @brief Функция расчёта первого элемента последовательности.

\* @param x - аргумент функции.

\* @return Возвращает значение первого элемента последовательности.

\*/

double GetFirstElement(int x);

/\*\*

\* @brief Точка входа в программу.

\* @return Возвращает 0 в случае успеха.

\*/

int main()

{

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

const double beginning\_interval = GetValue("Enter the beginning of the interval: ");

const double end\_interval = GetValue("Enter the end of the interval: ");

CheckSegment(beginning\_interval, end\_interval);

const double step = GetValue("Enter the function step: ");

CheckStep(step);

double x = beginning\_interval;

while (x - end\_interval <= DBL\_EPSILON)

{

printf\_s("%10.2lf | %25.15lf | %.15lf \n", x, GetFunction(x), GetSeriesSum(x, e));

x += step;

}

return 0;

}

double GetSeriesSum(int x, double e)

{

double current = 1;

double sum = current;

int n = 1;

while (fabs(current) > e)

{

current \*= GetRecurrent(x, n);

sum += current;

n++;

}

return sum;

}

double GetRecurrent(int x, double n)

{

return pow(x, 4) / ((4 \* n + 1) \* (4 \* n + 5));

}

double GetFunction(int x)

{

return 0.25 \* log((1 + x) / (1 - x)) + 0.5 \* atan(x);

}

void CheckStep(double step)

{

if (step < DBL\_EPSILON)

{

puts("The value was entered incorrectly!");

abort();

}

}

void CheckSegment(const double beginning\_interval, const double end\_interval)

{

if (beginning\_interval - end\_interval > DBL\_EPSILON)

{

puts("The interval is entered incorrectly!");

abort();

}

}

double GetValue(const char\* message)

{

double value;

printf\_s("%s", message);

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

if (result != 1)

{

errno = EIO;

perror("Input error");

abort();

}

return value;

}

double GetFirstElement(int x)

{

double current = pow(x, 5) / 5;

return current;

}

* 1. Решение тестового примера

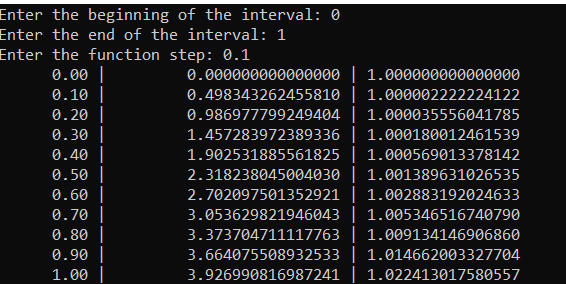


Рисунок 34 – Решение тестового примера

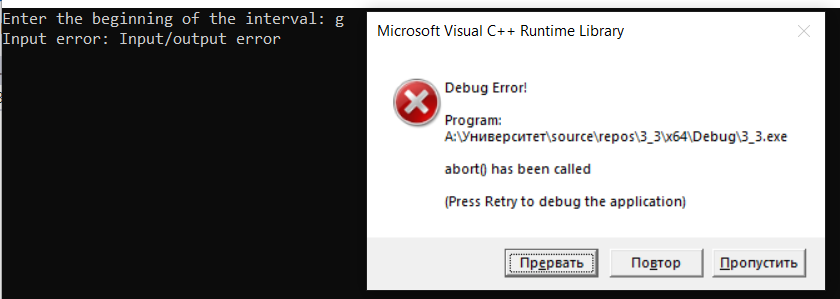


Рисунок 35 – Вывод программы, когда начало отрезка – буква

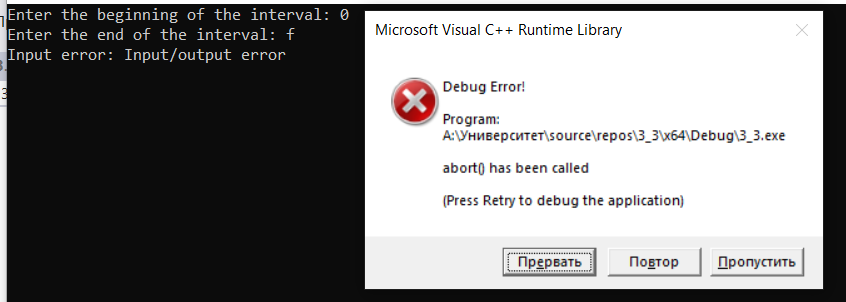


Рисунок 36 – Вывод программы, когда конец отрезка – буква

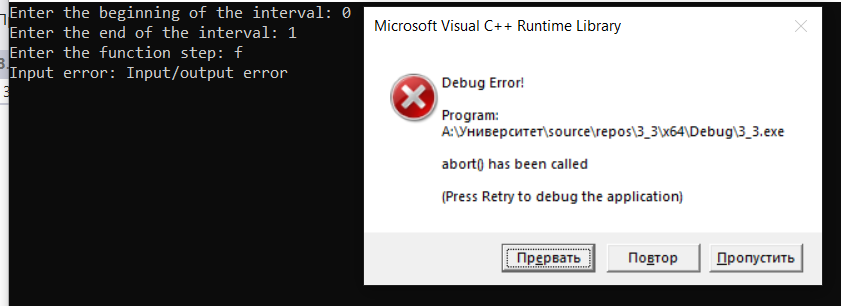


Рисунок 37 – Вывод программы, когда шаг функции – буква

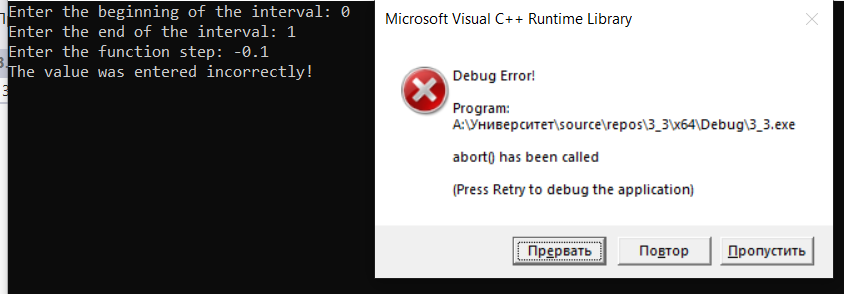


Рисунок 38 – Вывод программы, когда шаг функции отрицательный

* 1. Зачёт задания в GitHub

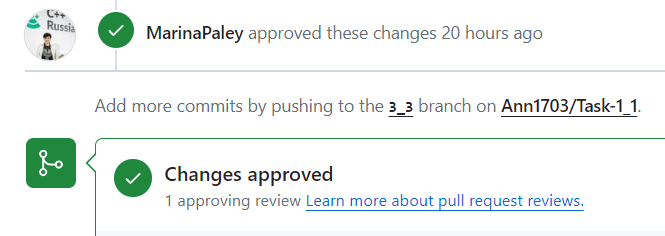


Рисунок 39 – Зачёт задания в GitHub