Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**«Вычисление косинуса, синуса, экспоненты**

**и натурального логарифма»**

**Выполнила**:

студентка группы 3821Б1ПМ2

Миронова А. А.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В. Д.

Нижний Новгород

2022

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 7](#_Toc26962564)

[Описание программной реализации 8](#_Toc26962565)

[Подтверждение корректности 9](#_Toc26962566)

[Результаты экспериментов 10](#_Toc26962567)

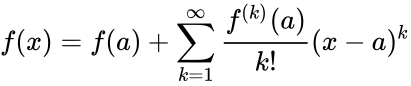
[Заключение 12](#_Toc26962568)

# Постановка задачи

Целью лабораторной работы являлась реализовать математические функции: синус, косинус, экспонента и натуральный логарифм в точке с помощью разложения в ряд Тейлора. Реализовать методы прямого, попарного и обратного суммирования элементов разложения. Нужно описать реализацию и алгоритмы работы программы, использованные в коде, подтвердить корректность результата функций, вычислить относительную погрешность. Описать способы подтверждения корректности, сделать вывод.

# Метод решения

Данное задание выполняем с помощью частного случая для рядов Тейлора – рядов Маклорена (при = 0). Формула ряда Тейлора получается из полинома степени n, который представляется в виде суммы по k = 0 до n коэффициентов , умноженных на . Последовательно берем 1-ю, 2-ю и так далее производные и подставляем точку вместо x. Получаем, что коэффициенты полинома выражаются через производную по номеру коэффициента, взятую в точке . Отсюда получаем формулу Тейлора алгебраического полинома



Эти формулы нужны для того, чтобы заменить данные нам функции на полиномы, которые будут с каждым слагаемым все точнее приближаться к исходным функциям, этот процесс называется аппроксимацией функции многочленами.

Формулы Маклорена:

* Экспонента



* Синус



* Косинус



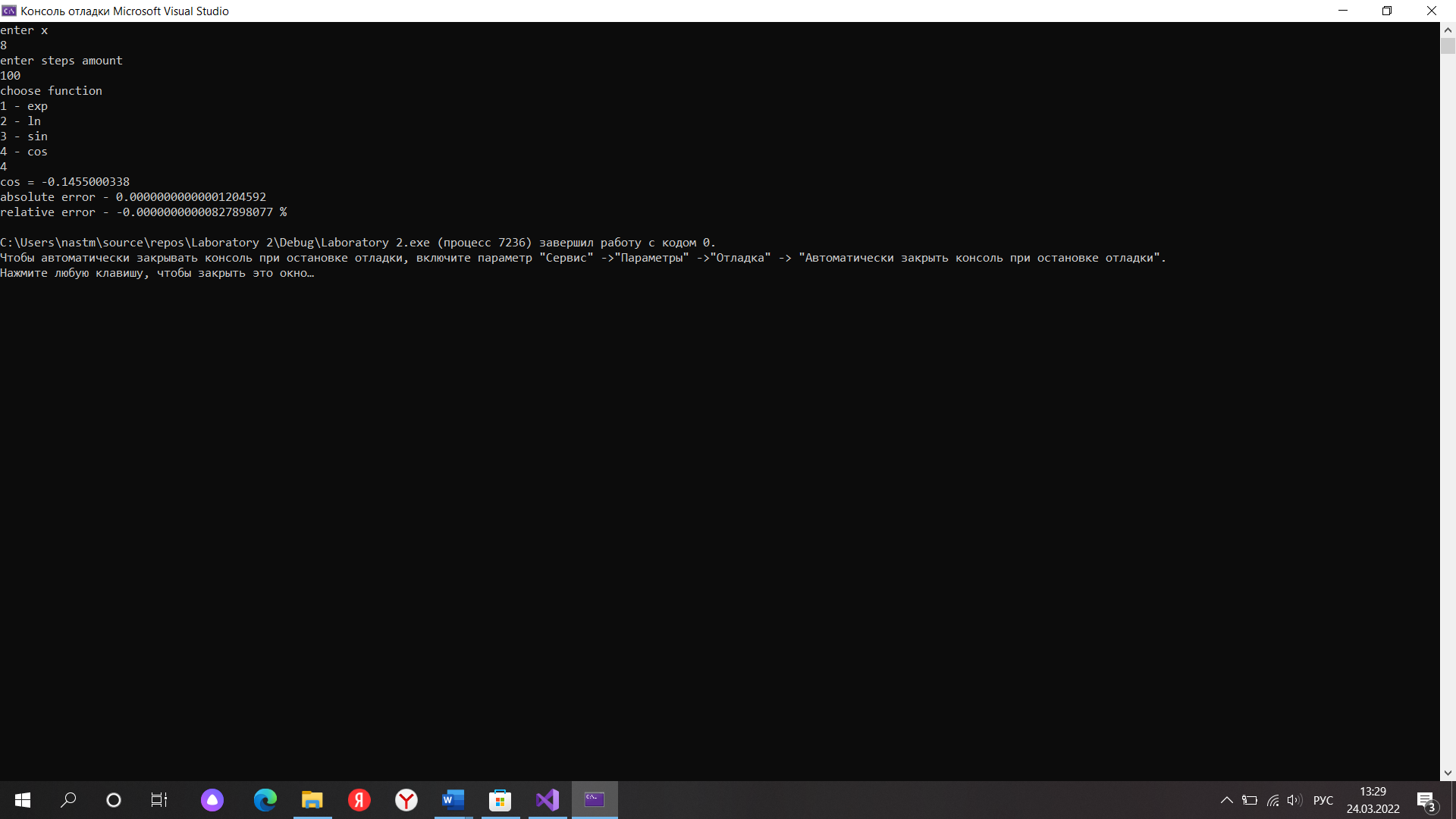
* Натуральный логарифм



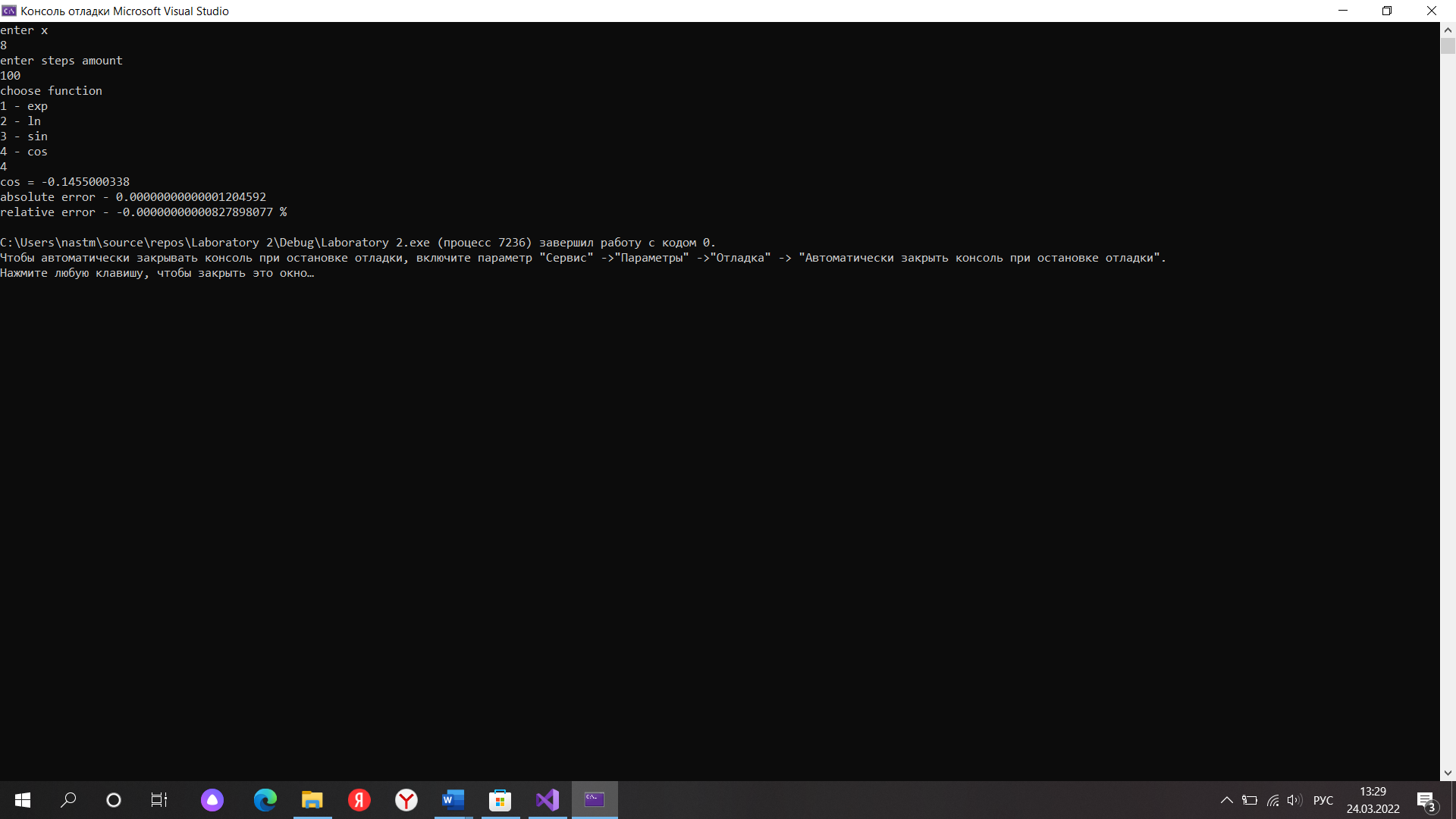
Для подсчета суммы будем использовать три способа суммирования: прямое суммирование, обратное суммирование и попарное суммирование.

# Руководство пользователя

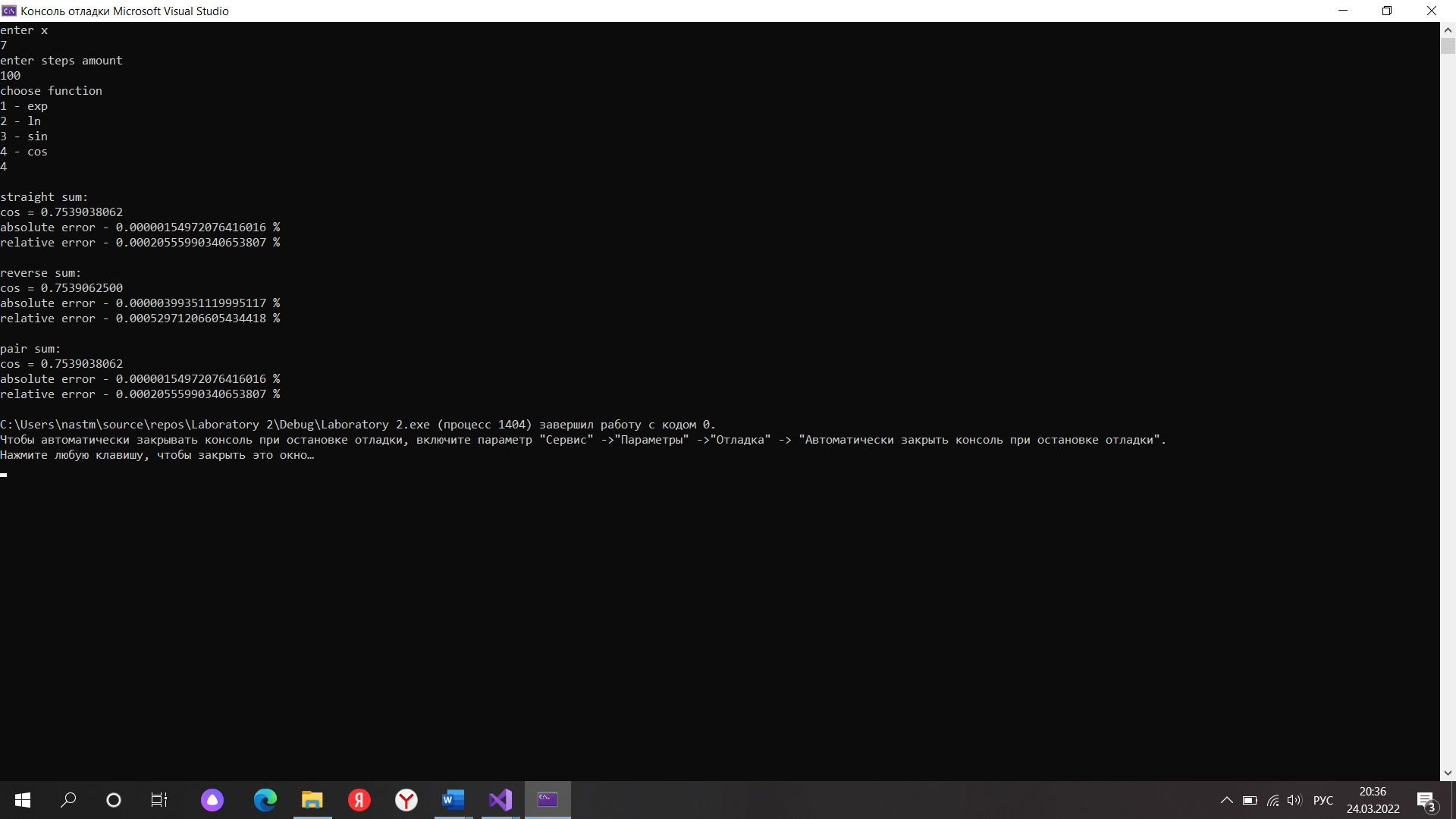
1. При запуске программы пользователь должен ввести X:



1. Потом перед пользователем появится выбор функции, для которой будет посчитана сумма:



1. После этого программа выведет результаты, который были получены прямым, обратным и попарным суммированиями, а также абсолютную и относительную ошибки:



# Описание программной реализации

**Используемые функции:**

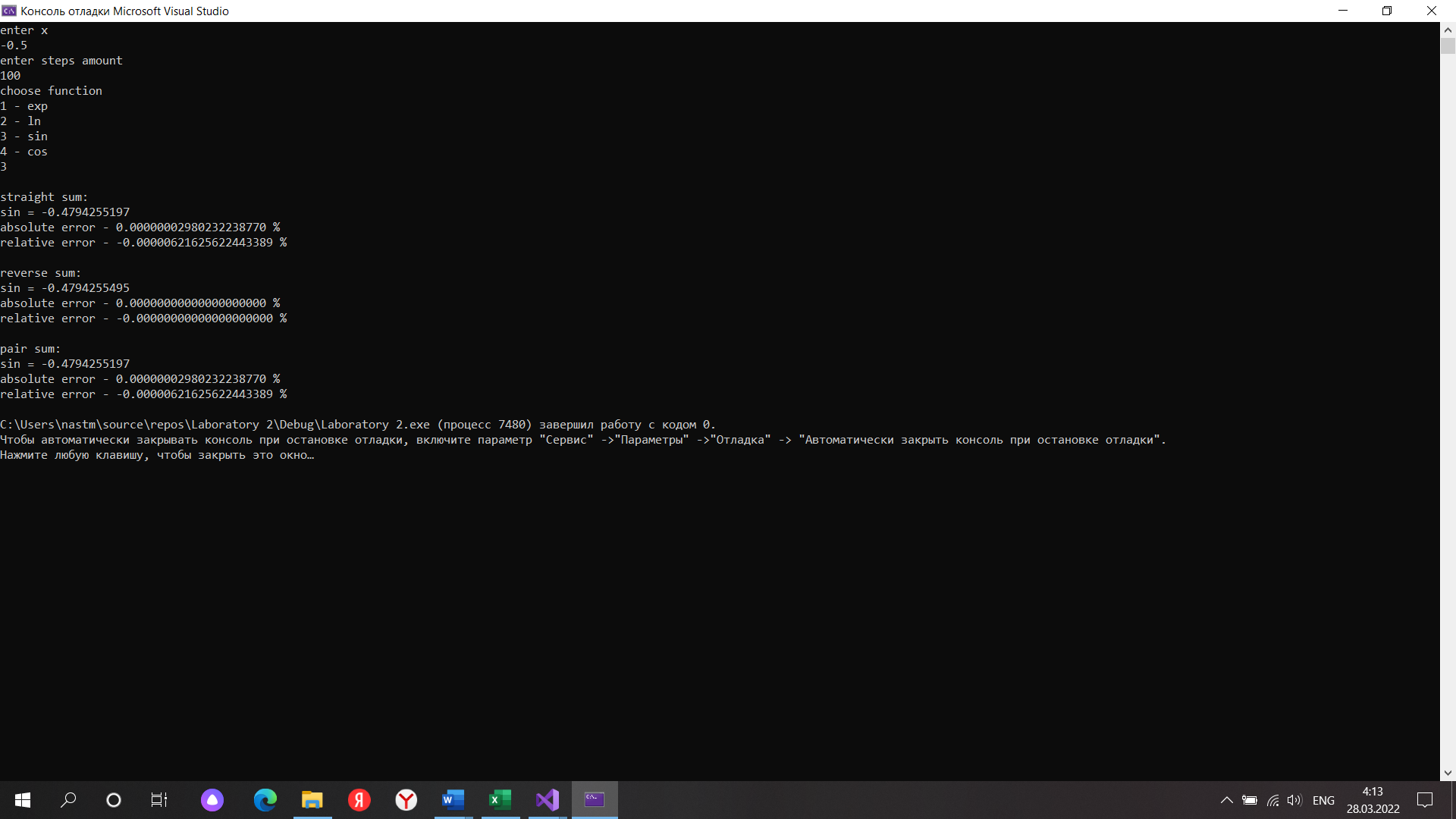
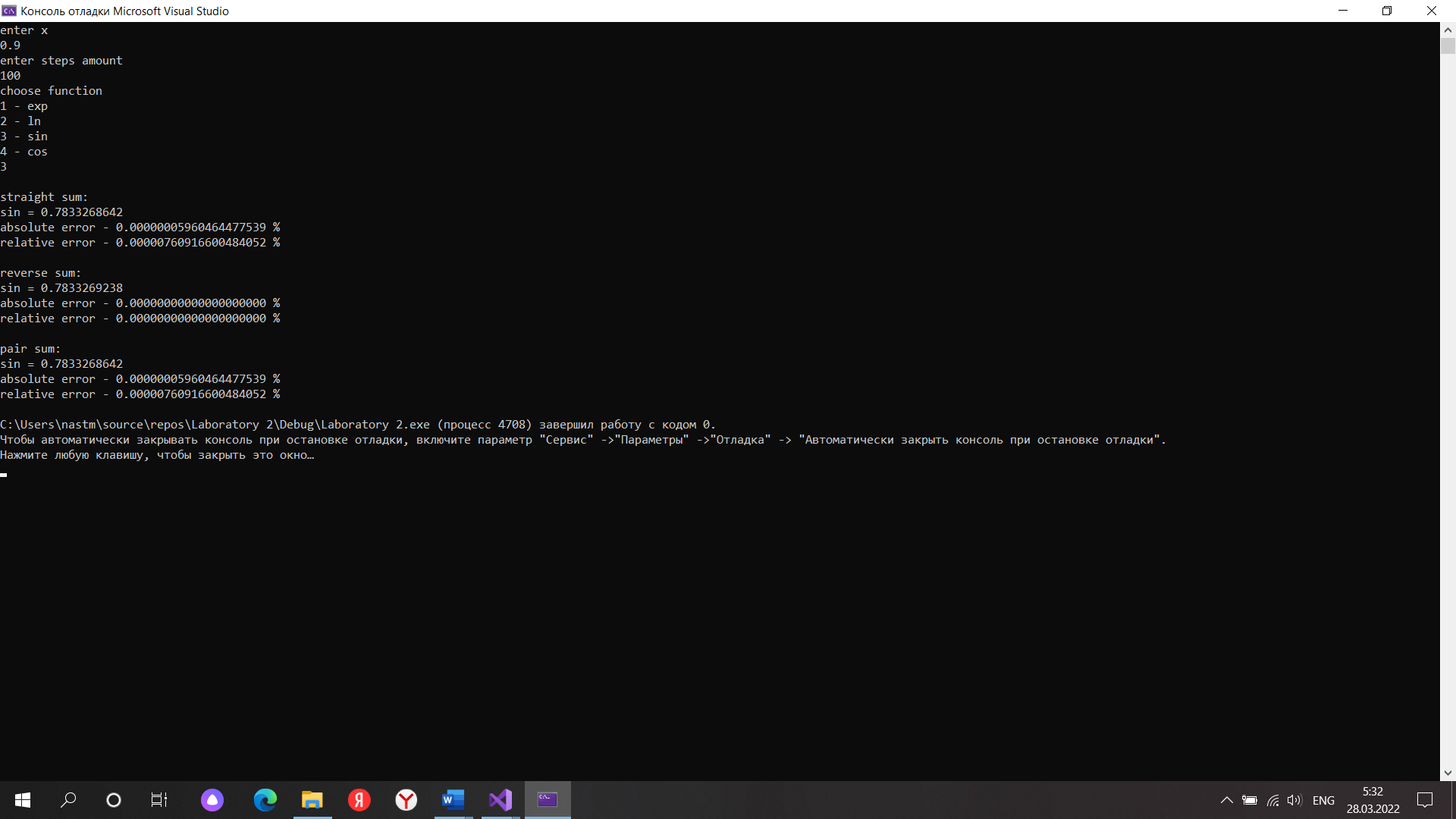
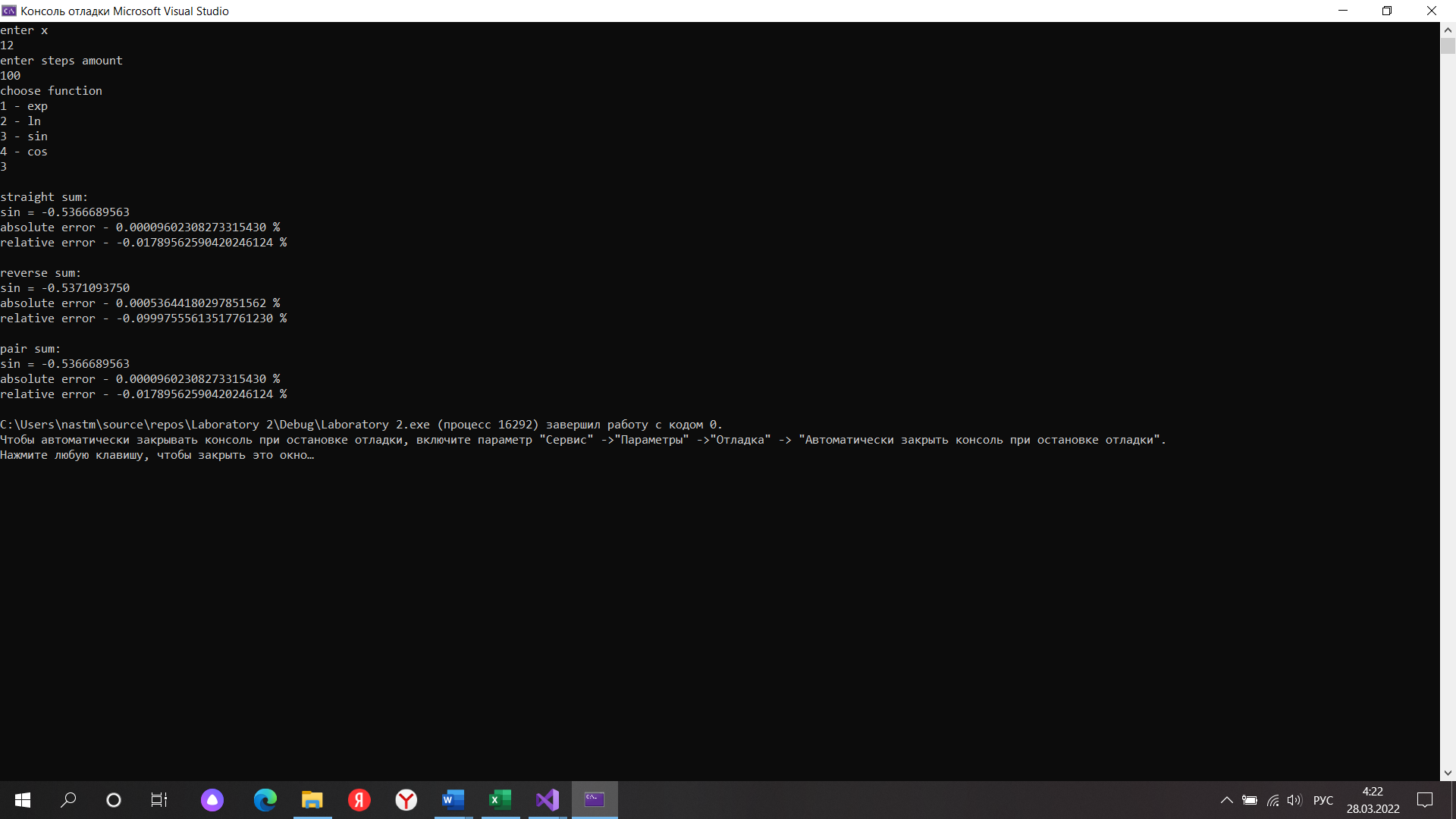
1. float exp\_next(float x, float prev, int n) – функция вычисления экспоненты;
2. float ln\_next(float x, float prev, int n) – функция вычисления натурального логарифма;
3. float sin\_next(float x, float prev, int n) – функция вычисления синуса;
4. float cos\_next(float x, float prev, int n) – функция вычисления косинуса;
5. float sumarray(float\* array, int n) – прямое суммирование реализовано в функции как обычная сумма n слагаемых;
6. float rev\_sumarray(float\* array, int n) – обратное суммирование – сумма n слагаемых в обратном цикле, начиная с последнего;
7. float pair\_sumarray(float\* array, int n) – попарное суммирование – складываем члены последовательности попарно, значения записываем в новую последовательность, а количество слагаемых при этом уменьшается вдвое;
8. void error\_check(double res, double(\*original)(double x), double x) – проверка на ошибки.

# Подтверждение корректности

Корректность исполнения программы проверяется за счет стандартных функций библиотеки <math.h>, а также во время подсчета относительной и абсолютной ошибок, где можно увидеть точность подсчета.

# Результаты экспериментов

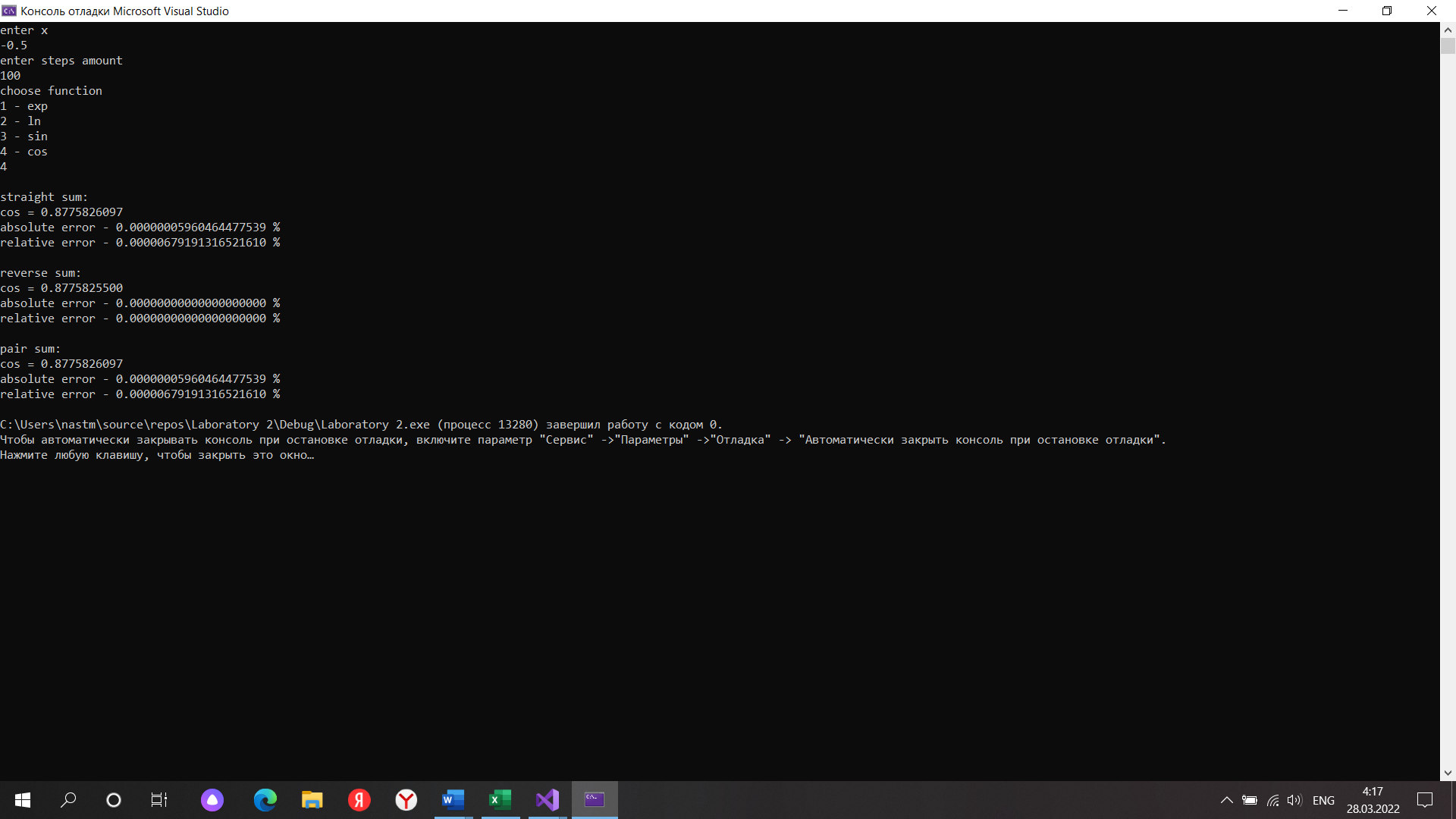
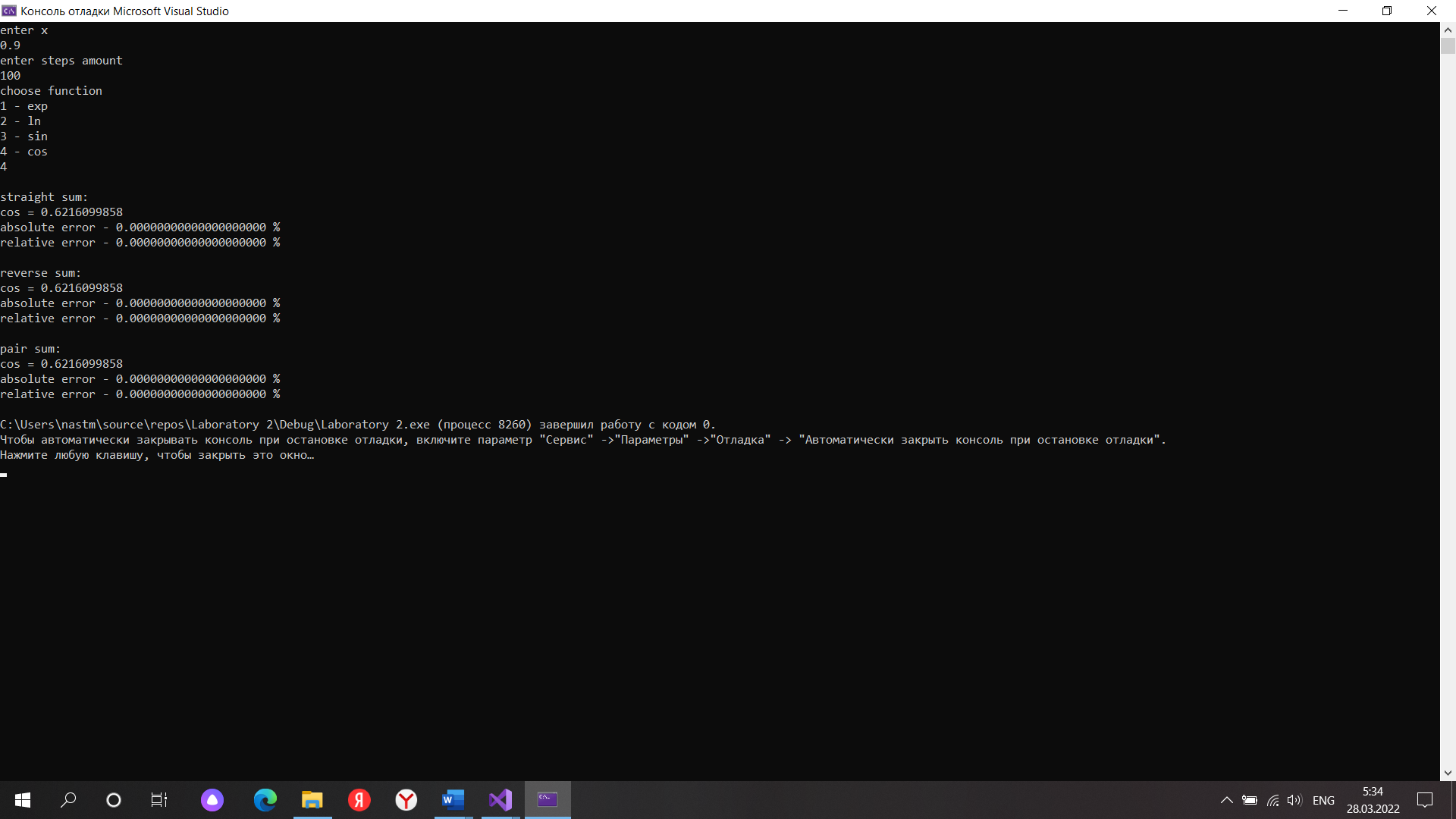
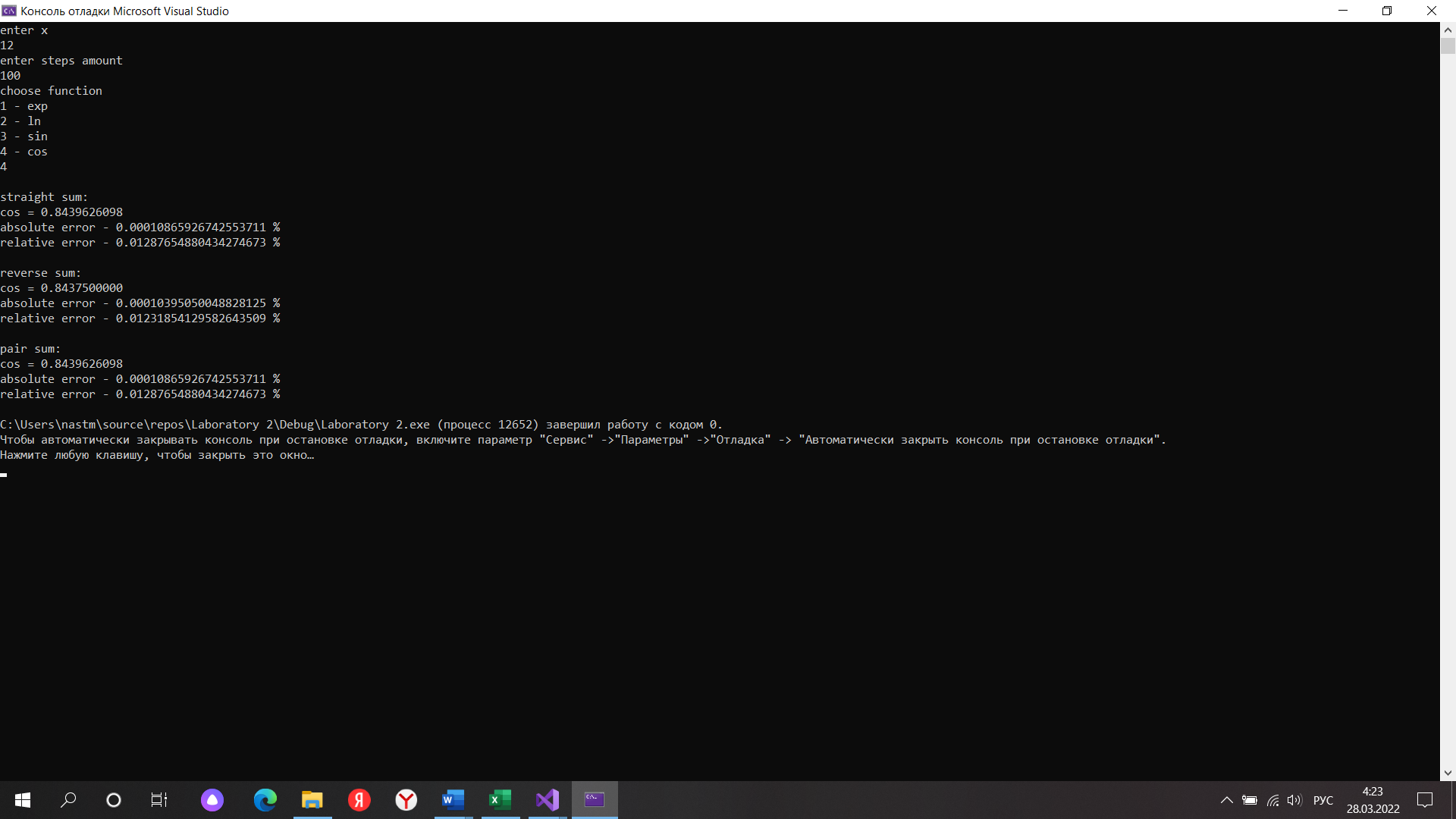
Программа считает относительную и абсолютную погрешность вычисления, поэтому в качестве эксперимента рассмотрим три (-0,5; 0,9; 12) случая для каждой функции:

Sin (x):

X = 12

X = 0,9

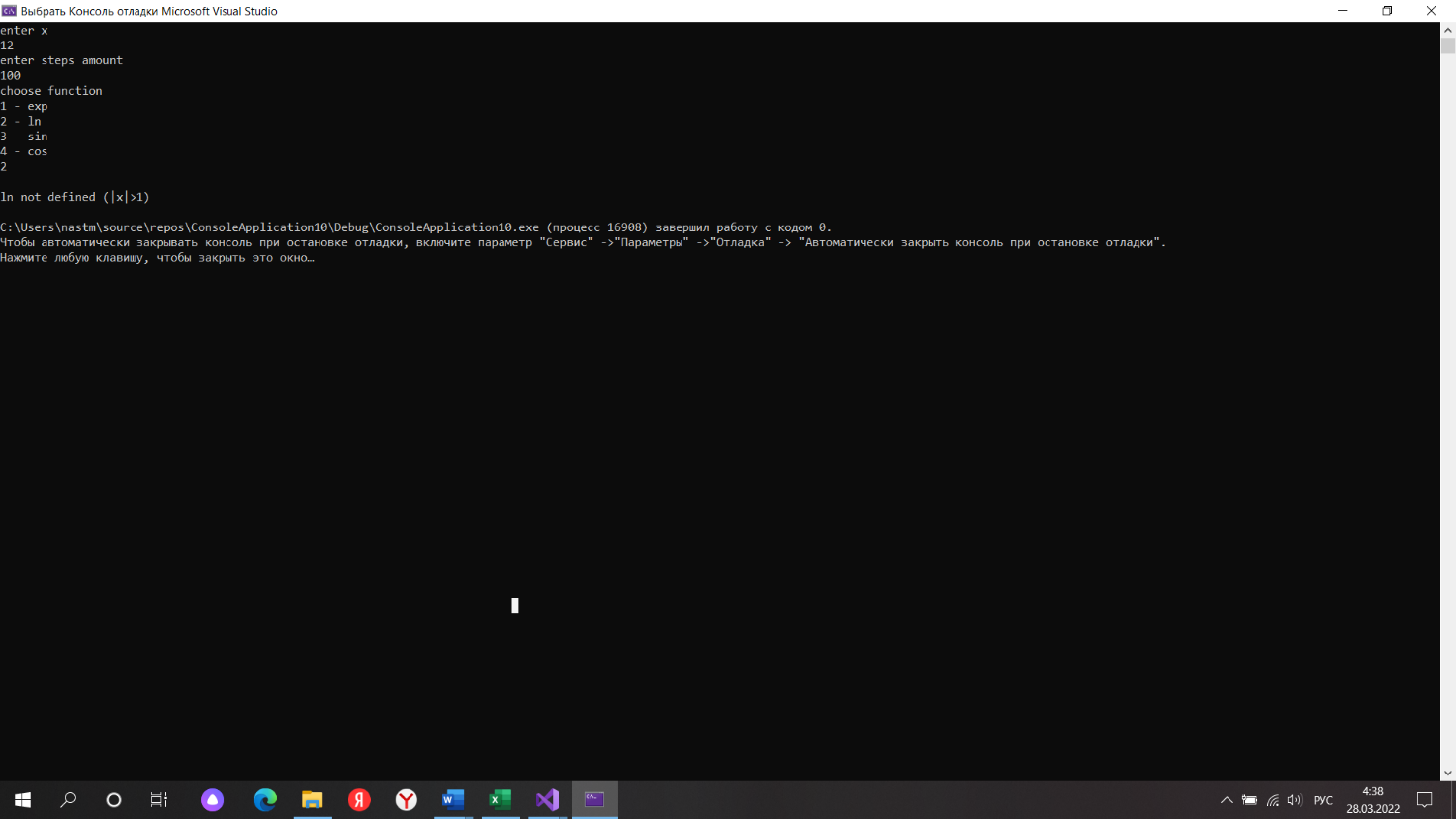
X = - 0,5

Cos(x):

X = 12

X = 0,9

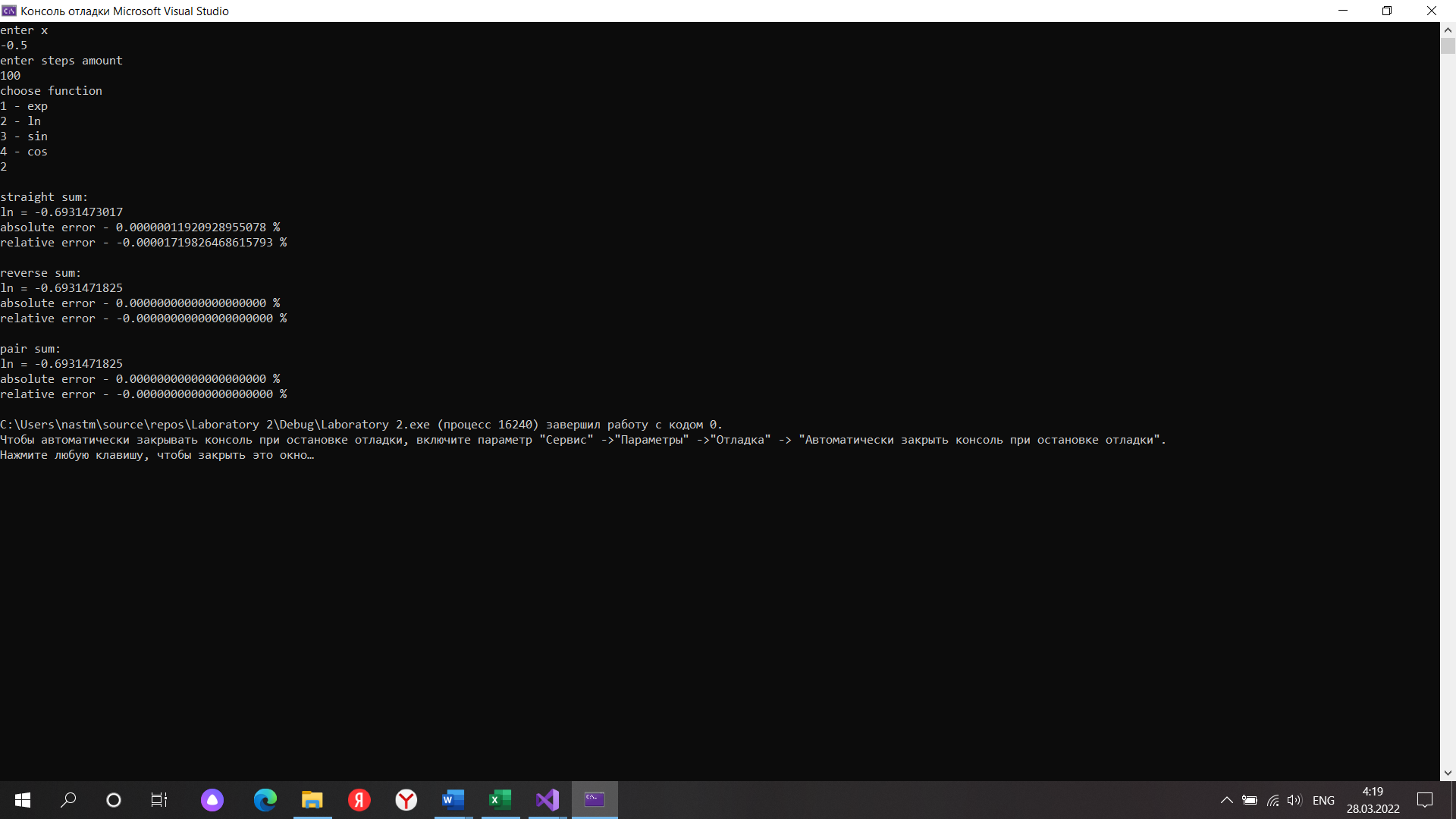
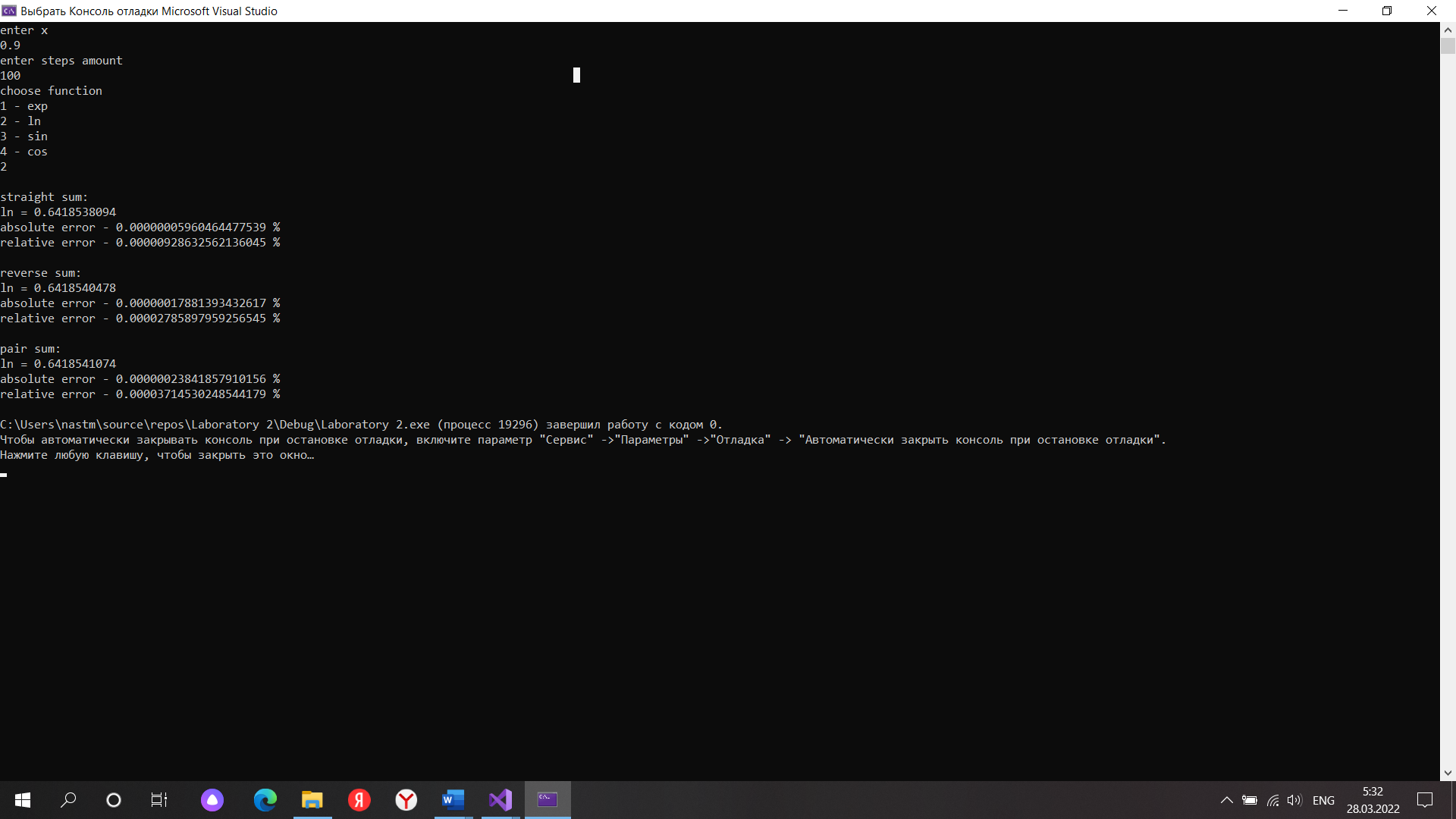
X = -0.5

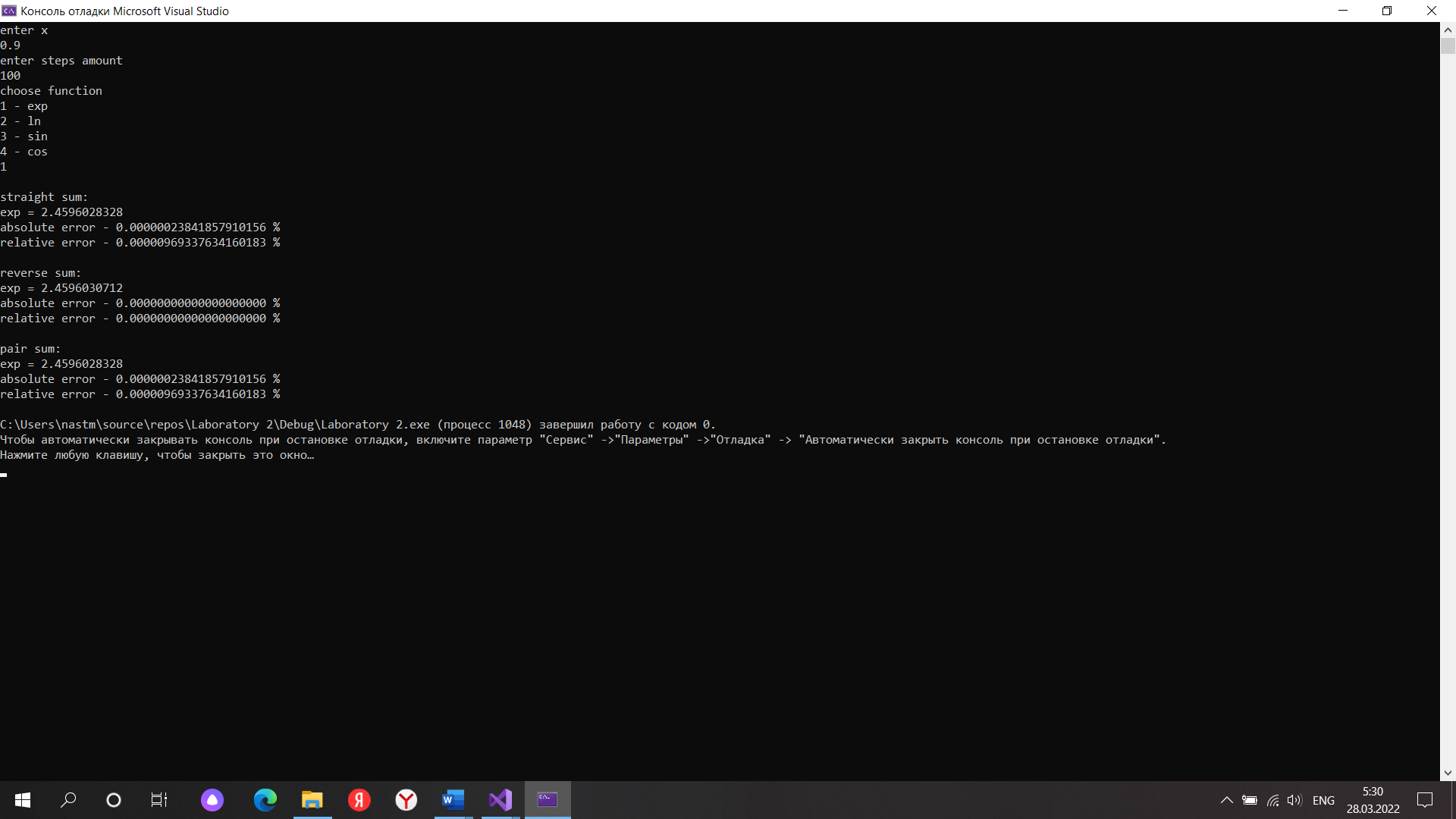
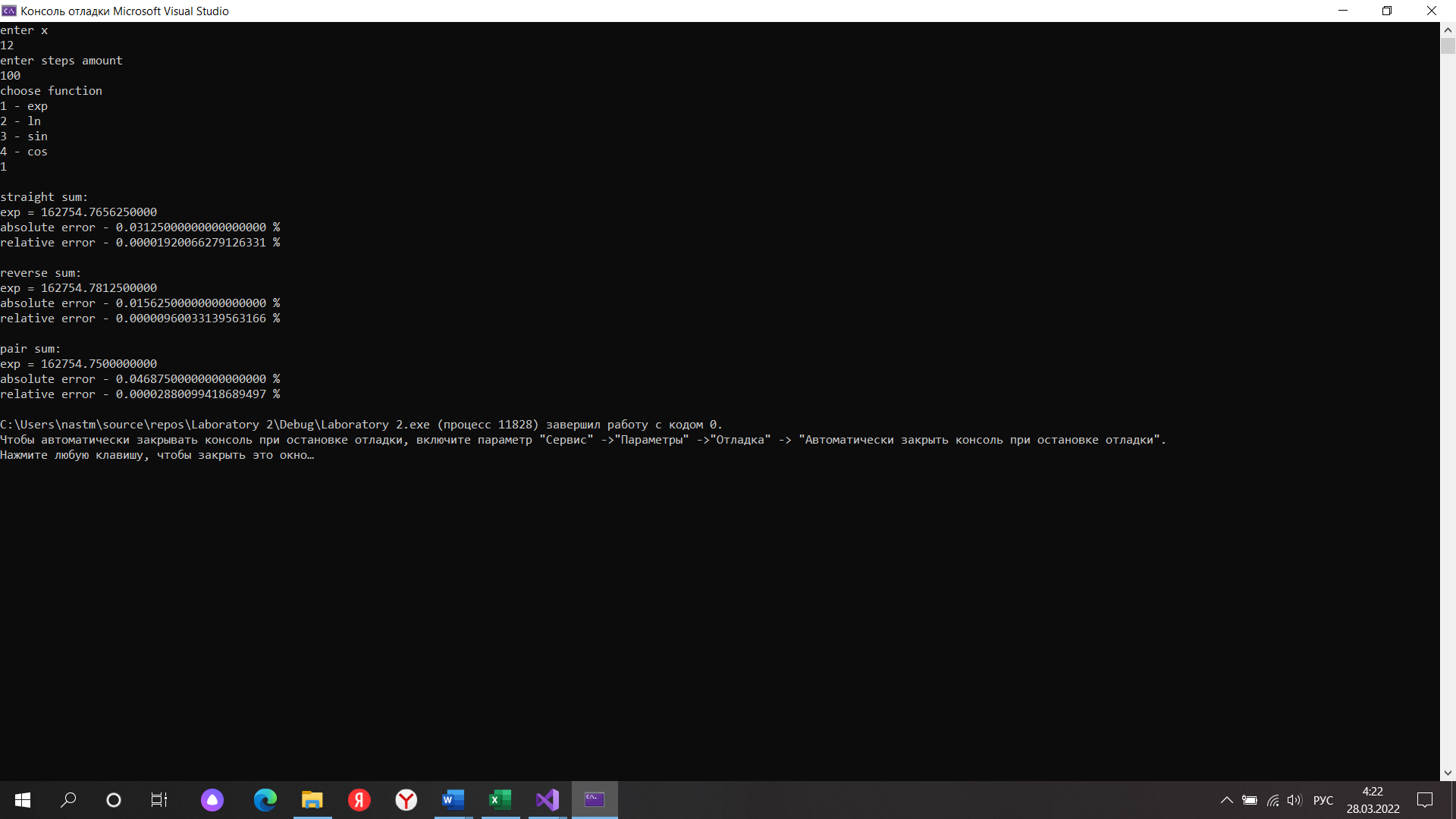
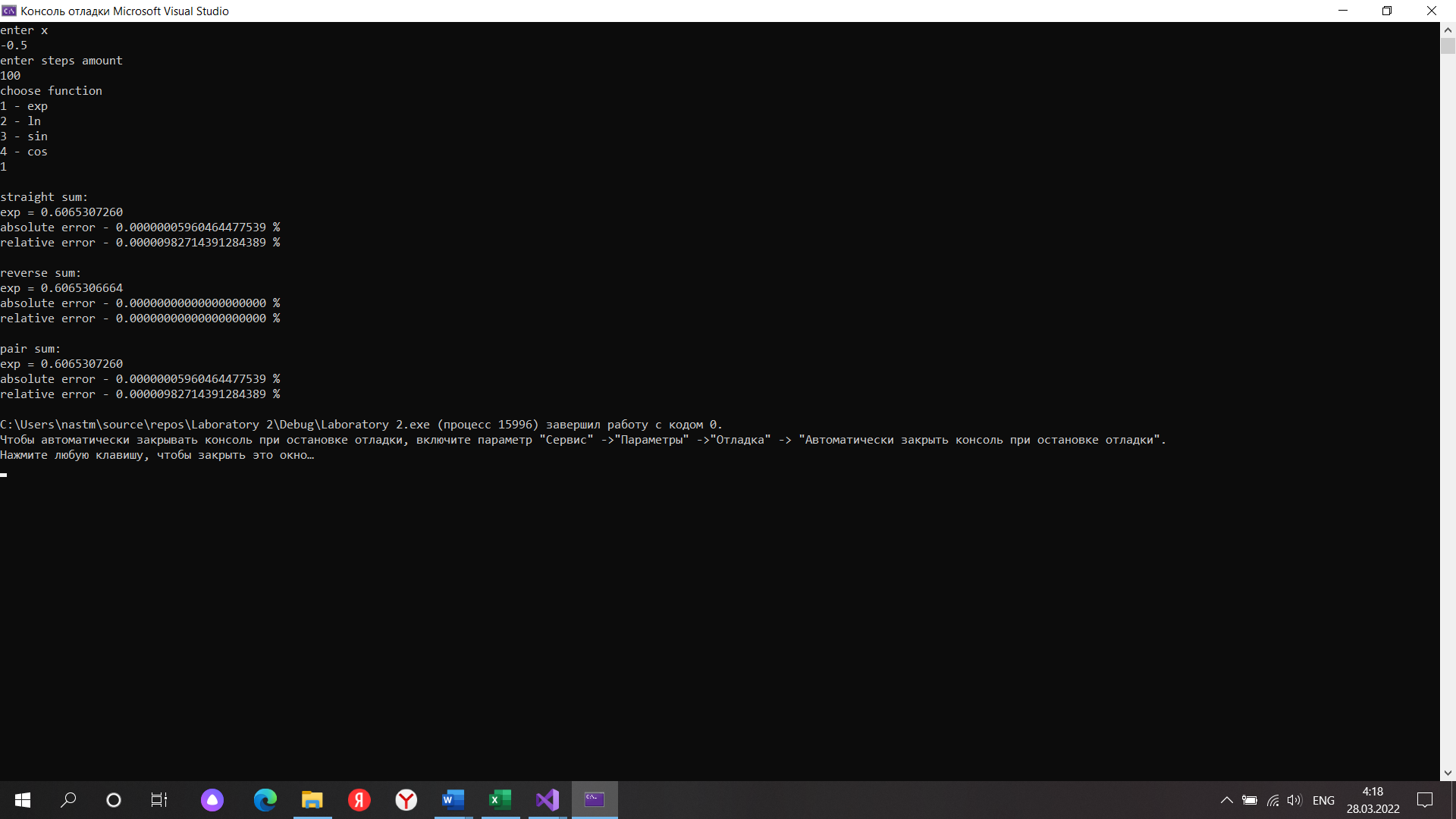
Ln(1+x):

X = -0,5

X = 12

X = 0,9



ex:

X = 12

X = 0,9

X = -0,5

Из результатов эксперимента получается, что программа корректно учитывает область определения функции Ln(1+x) и считает с допустимой неточностью значения остальных функций.

# Заключение

В ходе лабораторной работы был реализован алгоритм поиска значений sin, cos, ln(1+x), ex с помощью рядов Тейлора и алгоритм подсчета прямой, обратной и попарной суммы для этих функций.

По полученным данным входе работы, можно сделать вывод, что с увеличение задаваемого числа X, ошибка программы увеличивается. В случае же с логарифмом – ошибка тем будет больше, чем ближе число к границам области определения функции. Также в результате экспериментов получается, что на значения косинуса и синуса не влияет то, каким именно образом будет осуществляться суммирование. А значения экспоненты будет точнее при прямом суммировании, а у логарифма – при обратном.

# Приложение

float exp\_next(float x, float prev, int n) {

return (prev \* x) / (float(n));

}

float ln\_next(float x, float prev, int n) {

float res = -((prev \* float(n - 1) \* x) / float(n));

//printf("%lf ;%lf;%lf;%d \n", res,prev,x,n);

return res;

}

float sin\_next(float x, float prev, int n) {

if (n % 2 == 1) {

float res = (0 - 1) \* (prev \* x \* x) / (float(n \* (n - 1)));

return res;

}

else {

return 0;

}

}

float cos\_next(float x, float prev, int n) {

if (n % 2 == 1) {

return 0;

}

else {

float res = (0 - 1) \* (prev \* x \* x) / (float(n \* (n - 1)));

return res;

}

}

void printarray(float\* array, int n) {

for (int i = 0; i < n; i++) {

printf("%.20f \n", array[i]);

}

}

float sumarray(float\* array, int n) {

float sum = 0;

for (int i = 0; i < n; i++) {

sum += array[i];

}

return sum;

}

float rev\_sumarray(float\* array, int n) {

float sum = 0;

for (int i = n - 1; i >= 0; i--) {

sum += array[i];

}

return sum;

}

float pair\_sumarray(float\* array, int n) {

float sum = 0;

for (int i = 0; i < n - 1; i += 2) {

sum += array[i] + array[i + 1];

}

return sum;

}

void error\_check(float res, float(\*original)(float x), float x) {

printf("absolute error - %.20lf %%\n", fabs(original(x) - res));

printf("relative error - %.20lf %% \n", fabs(original(x) - res) / original(x) \* 100);