/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Project name : Kursovaya \*

\* Project type : win32 console app \*

\* File name : Kursovaya \*

\* Programmers : Usupov Tengiz T. \*

\* Modified : - \*

\* Created : 10.12.18 11:10 \*

\* Last Revision : 12.12.18 17:55 \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define \_USE\_MATH\_DEFINES

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <math.h>

#include <stdio.h>

#include <ctype.h>

#include <float.h>

#include <errno.h>

#include <locale.h>

#include <string.h>

#define ex 12345678900//Значения для передачи функции

#define iks 123456789001//определяющие число/х/е

#define mines\_iks 1234567890010

#define mines\_ex 123456789002

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* П Р О Т О Т И П Ы Ф У Н К Ц И Й \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Функция для интегрирования \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double Function(double x);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Первоначальне данные о переменных \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Installation(char Massive[], int length);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Приведение к типу double \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Bring();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Метод трапеций \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Trapeze();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Метод прямоугольников \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Rectangle();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Подсчет приближенного значения(П) \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double ApproximationRectangle(int a);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Подсчет приближенного значения(Т) \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double ApproximationTrapeze(int a);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Переменные для считывания из массива\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

char digit\_1[3],

digit\_2[3],

trigonometric[6],

restrict\_1[5],

restrict\_2[5],

arg\_f[3],

constant[3];

double NUMB\_1, NUMB\_2, NUMB\_3 = 1;

double NUMB\_4, upper\_limit, lower\_limit;

char sign\_1, sign\_2,

flag\_1 = 0, flag\_2 = 0, flag\_3 = 0;//переменные и знаки приведенные к нормальному виду

int iteration = 0;//для подсчета итерраций

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* О С Н О В Н А Я П Р О Г Р А М М А \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

const char EndFile = 3;

const char NameFile[] = "OUT.bin";// исходный файл

char line[70];

int length;

int j = 0; // число строк

FILE \*in; // поток данных

in = fopen(NameFile, "wb");//открываем файл в двоичном виде

setlocale(LC\_ALL, "RUSSIAN");

if (in != NULL)// если файл открылся

{

for (fread(&length, 1, 1, in); length != EndFile; j++)// пока не достигнем конца файла

{

fread(line, length, 1, in);

length = 0;// обнуляем значения для новой строки

fread(&length, 1, 1, in);

}

fclose(in);// закрываем файл

}

else

{

printf("Ошибка открытия файла \"%s\"!\n", NameFile);// если что-то пошло не так — выводим ошибку

}

if (j == 1)// если получена только одна строка

{

Installation(line, length);//функия, выделяющая коэфициенты из нашего уравнения

Bring();//приведение переменных к типу double

Trapeze(NUMB\_1, NUMB\_2, NUMB\_3, NUMB\_4, lower\_limit, upper\_limit, sign\_1, sign\_2, trigonometric);

Rectangle(NUMB\_1, NUMB\_2, NUMB\_3, NUMB\_4, lower\_limit, upper\_limit, sign\_1, sign\_2, trigonometric);

}

\_getch();

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* РЕАЛИЗАЦИЯ ФУНКЦИЙ \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Installation(char Massive[], int length)

{

int pos\_step, end\_step, pos\_skobki, pos\_skobki2, end\_f, start\_ogr,

end\_ogr=0;

//получаем первую цифрy

for (int n = 0; n < length; n++)

{

if ((Massive[n] == 'y') && (Massive[n + 1] == '='))

{

if (isdigit(Massive[n + 2]) || (Massive[n + 2] == '-'))

{

for (int i = 0; i < 5; i++)// 5 - длина

{

if (Massive[n + 2 + i] == '^')

{

pos\_step = n + 2 + i;

break;

}//if

digit\_1[i] = Massive[n + 2 + i];

}//for

}//if

if (Massive[n + 2] == 'e')

{

flag\_1 = 1;

pos\_step = n + 3;

}//if

if (Massive[n + 2] == 'x')

{

flag\_1 = 2;

pos\_step = n + 3;

}//if

if ((Massive[n + 2] == '-') && (Massive[n + 3] == 'x'))

{

flag\_1 = 3;

pos\_step = n + 4;

}//if

if ((Massive[n + 2] == '-') && (Massive[n + 3] == 'e'))

{

flag\_1 = 4;

pos\_step = n + 4;

}//if

//получаем степень числа

if (isdigit(Massive[pos\_step + 1]) || (Massive[pos\_step + 1] == '-'))

{

for (int i = 0; i < 5; i++)// 5 - длина

{

if (!isdigit(Massive[pos\_step + 1 + i]) && (Massive[pos\_step + i] != '^') && (Massive[pos\_step + i] != '-'))

{

end\_step = pos\_step + 1 + i;

break;

}//if

digit\_2[i] = Massive[pos\_step + 1 + i]; //степень

}//for

}//if

if (Massive[pos\_step + 1] == 'e')

{

flag\_2 = 1;

end\_step = pos\_step + 2;

}//if

if (Massive[pos\_step + 1] == 'x')

{

flag\_2 = 2;

end\_step = pos\_step + 2;

}//if

if ((Massive[pos\_step + 1] == '-') && (Massive[pos\_step + 2] == 'x'))

{

flag\_2 = 3;

end\_step = pos\_step + 3;

}//if

n = end\_step;

sign\_1 = Massive[n]; //первый знак

//получаем тригонометрическую функцию

for (int i = 0; i < 7; i++)

{

if (Massive[n + 1 + i] == '(')

{

pos\_skobki = n + 1 + i;

break;

}//if

trigonometric[i] = Massive[n + 1 + i]; //тригонометрич ф-я

}//for

//получаем аргумент

for (int i = 0; i < 5; i++)// 5 - длина аргумента триг ф.

{

if (!isdigit(Massive[pos\_skobki + 1]))

{

pos\_skobki2 = pos\_skobki + 2;

flag\_3 = 1;

break;

}//if

else

if (Massive[pos\_skobki + 1 + i] == '\*')

{

pos\_skobki2 = pos\_skobki + i + 3;

break;

}//if

else arg\_f[i] = Massive[pos\_skobki + 1 + i]; //аргумент функции

}//for

sign\_2 = Massive[pos\_skobki2 + 1]; //второй знак

//получаем константу

for (int i = 0; i < 5; i++)// 5 - длина

{

if (Massive[pos\_skobki2 + 2 + i] == ',')

{

end\_f = pos\_skobki2 + 2 + i;

break;

}//if

constant[i] = Massive[pos\_skobki2 + 2 + i]; //запись числа в пер.

}//for

//получаем границы интегрирования

for (int i = 0; i < length; i++)

{

if (Massive[end\_f + i] == 'x')

{

start\_ogr = end\_f + i + 3; // xэ(\_, )

for (int i = 0; i < 5; i++)

{

if (Massive[start\_ogr + i] == ',')

{

end\_ogr = start\_ogr + i;

break;

}//if

restrict\_1[i] = Massive[start\_ogr + i];

}//for

for (int i = 0; i < 5; i++) // xэ( ,\_)

{

if (Massive[end\_ogr + 1 + i] == ')')

break;

restrict\_2[i] = Massive[end\_ogr + 1 + i];

}//for

}//if

}//for

}//if

break;

}//for

}//Installation

void Bring()

{

if (flag\_1 == 0) NUMB\_1 = atof(digit\_1);

if (flag\_1 == 1) NUMB\_1 = ex;

if (flag\_1 == 2) NUMB\_1 = iks;

if (flag\_1 == 3) NUMB\_1 = mines\_iks;

if (flag\_1 == 4) NUMB\_1 = mines\_ex;

if (flag\_2 == 0) NUMB\_2 = atof(digit\_2);

if (flag\_2 == 1) NUMB\_2 = ex;

if (flag\_2 == 2) NUMB\_2 = iks;

if (flag\_2 == 3) NUMB\_2 = mines\_iks;

if (flag\_3 == 0) NUMB\_3 = atof(arg\_f);

NUMB\_4 = atof(constant);

lower\_limit = atof(restrict\_1);

upper\_limit = atof(restrict\_2);

}//Bring

double Function(double x)

{

double a, b, f;

if (NUMB\_1 == ex)

a = exp(NUMB\_2);

if (NUMB\_1 == iks)

a = pow(x, NUMB\_2);

if (NUMB\_1 == mines\_iks)

a = pow(-x, NUMB\_2);

if (NUMB\_1 == mines\_ex)

a = -exp(NUMB\_2);

if ((NUMB\_1 == 0) && (NUMB\_2 < 0))

NUMB\_1 = DBL\_EPSILON;

if (NUMB\_2 == ex)

a = pow(NUMB\_1, exp(1));

if (NUMB\_2 == iks)

a = pow(NUMB\_1, x);

if (NUMB\_2 == mines\_iks)

a = pow(NUMB\_1, -x);

if ((NUMB\_1 != ex) && (NUMB\_1 != iks) && (NUMB\_1 != mines\_iks)

&& (NUMB\_2 != ex) && (NUMB\_2 != iks) && (NUMB\_2 != mines\_ex))

a = pow(NUMB\_1, NUMB\_2);

if ((trigonometric[0] == 'c') && (trigonometric[2] == 's'))

b = cos(NUMB\_3\*x);

if ((trigonometric[0] == 's') && (trigonometric[2] == 'n'))

b = sin(NUMB\_3\*x);

if ((trigonometric[0] == 't') && (trigonometric[1] == 'g'))

b = tan(NUMB\_3\*x);

if ((trigonometric[0] == 'c') && (trigonometric[2] == 'g'))

b = (NUMB\_3\*x) / sin(NUMB\_3\*x);//ctg

if ((trigonometric[0] == 'a') && (trigonometric[1] == 's') && (trigonometric[3] == 'n'))

b = asin(NUMB\_3\*x);

if ((trigonometric[0] == 'a') && (trigonometric[3] == 's'))

b = acos(NUMB\_3\*x);

if ((trigonometric[0] == 'a') && (trigonometric[4] == 'g'))

b = atan(NUMB\_3\*x);

if ((trigonometric[0] == 's') && (trigonometric[1] == 'h'))

b = sinh(NUMB\_3\*x);

if ((trigonometric[0] == 'c') && (trigonometric[1] == 'h'))

b = cosh(NUMB\_3\*x);

if ((trigonometric[0] == 't') && (trigonometric[1] == 'h'))

b = tanh(NUMB\_3\*x);

if (sign\_1 == '+')

f = a + b;

if (sign\_1 == '-')

f = a - b;

if (sign\_1 == '\*')

f = a \* b;

if (sign\_1 == '/')

{

if (fabs(b) < DBL\_EPSILON)

b = DBL\_EPSILON;

f = a / b;;

}//if

if (sign\_2 == '+')

f = f + NUMB\_4;

if (sign\_2 == '-')

f = f - NUMB\_4;

if (sign\_2 == '\*')

f = f \* NUMB\_4;

if (sign\_2 == '/')

{

if (fabs(NUMB\_4) < DBL\_EPSILON)

b = DBL\_EPSILON;

f = f / NUMB\_4;

}//if

iteration++;

return f;

}//Function

void Trapeze()

{

iteration = 0;

double eps = 0.0005;//точность

double res1;//результаты

double res;

int N = 10;//начальное число отрезков разбиения

int k = 5;//на сколько увеличить число отрезков разбиения

int i = 0;

res1 = ApproximationTrapeze(N);

do

{

res = res1;

N = N + k;

res1 = ApproximationTrapeze(N);

i++;

} while (fabs(res1 - res) > eps);

printf("Точное значение(метод ТРАПЕЦИЙ) : %f\n", res1);

printf("Итераций = %d, i= %d\n", iteration, i); //кол-во вычислений функции

}//Trapeze

void Rectangle()

{

iteration = 0;

double eps = 0.0005;//точность

double res1;

double res;

int N = 10;//начальное число отрезков разбиения

int k = 5;//на сколько увеличить число отрезков разбиения

int i = 0;

res1 = ApproximationRectangle(N);

do

{

res = res1;

N = N + k;

res1 = ApproximationRectangle(N);

i++;

} while (fabs(res1 - res) > eps);

printf("Точное значение(метод ПРЯМОУГОЛЬНИКОВ): %f\n", res1);

printf("Итераций = %d, i= %d\n", iteration, i);

}//Rectangle

double ApproximationTrapeze(int a)

{

double sum = 0.0;

double H = ((upper\_limit - lower\_limit) / (1.0 \* a));

for (int i = 1; i < a; i++)

{

errno = Function(lower\_limit + H\*i) \* H;

if ((errno == EDOM) || (errno == HUGE\_VAL))

{

sum += (Function(lower\_limit + H\*i + DBL\_EPSILON) \* H);

}//if

else

{

sum += Function(lower\_limit + H\*i) \* H;

}//else

}//for

sum = sum + (((Function(lower\_limit) + Function(upper\_limit)) / 2.0) \* H);

return sum;

}//ApproximationTrapeze

double ApproximationRectangle(int a)

{

double sum = 0.0;

double H = ((upper\_limit - lower\_limit) / (1.0 \* a));

for (int i = 1; i < a; i++)

{

errno = Function(lower\_limit + H\*i + H / 2.0);

if ((errno == EDOM) || (errno == HUGE\_VAL))

{

sum += Function(lower\_limit + H\*i + DBL\_EPSILON + H / 2.0);

}//if

else

{

sum += Function(lower\_limit + H\*i + H / 2.0);

}//else

}//for

return (sum \* H);

}//ApproximationRectangle