**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе №5**

**по дисциплине «Основы искусственного интеллекта»**

Тема: Алгоритм Пчелиная колония

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9302 |  | Ширнин К.В. |
|  |  | Квитко Д.В. |
| Преподаватель |  | Новакова Н.Е. |

Санкт-Петербург

2022

**Содержание**

[Цель работы 3](#_Toc121440160)

[UML-диаграмма 4](#_Toc121440161)

[Описание классов 5](#_Toc121440162)

[Ход работы 7](#_Toc121440163)

[Пример работы программы 8](#_Toc121440164)

[Вывод 9](#_Toc121440165)

[Список используемой литературы 9](#_Toc121440166)

[Приложение 10](#_Toc121440167)

# Цель работы

Изучить принцип работы пчелиного алгоритма. Написать ПО с пользовательским графическим интерфейсом, находящее минимум заданной функции, с использованием пчелиного алгоритма

**Теор****ия**

Алгоритм пчелиной колонии (алгоритм оптимизации подражанием пчелиной колонии, англ. artificial bee colony optimization, ABC) — один из полиномиальных эвристических алгоритмов для решения оптимизационных задач в области информатики и исследования операций. Относится к категории стохастических бионических алгоритмов, основан на имитации поведения колонии медоносных пчел при сборе нектара в природе.

Основной целью работы пчелиной колонии в природе является разведка пространства вокруг улья с целью поиска нектара с последующим его сбором. Для этого в составе колонии существуют различные типы пчел: пчелы-разведчики и рабочие пчелы-фуражиры (кроме них, в колонии существуют трутни и матка, не участвующие в процессе сбора нектара). Разведчики ведут исследование окружающего улей пространства и сообщают информацию о перспективных местах, в которых было обнаружено наибольшее количество нектара (для обмена информацией в улье существует специальный механизм, именуемый танцем пчелы).

Алгоритм пчелиной колонии может применяться для решения дискретных (комбинаторных) и непрерывных задач глобальной оптимизации. Обычно он включает в себя начальную разведку и последующую работу пчел улья. При инициализации (начальной разведке) производится выполнение разведки пространства признаков с целью определения его наиболее перспективных точек с наилучшими значениями целевой функции в простейшем случае с использованием метода случайного перебора), которые запоминаются в улье. После этого в окрестностях выбранных точек производится локальная разведка в пределах заданного радиуса разведки с целью попытки уточнения решения (улучшения рекорда), при этом при достижении улучшения в улье сохраняется обновленное значение рекорда и соответствующий ему вектор параметров целевой функции . Комбинируя работу пчел-разведчиков и рабочих пчел на протяжении заданного числа итераций C алгоритм обеспечивает постепенное улучшение запоминаемой выборки из решений. По завершении его работы из указанного множества решений выбирается наилучшее, что является результатом работы алгоритма.

# UML-диаграмма

На рисунке 1 представлена UML-диаграмма структуры программы

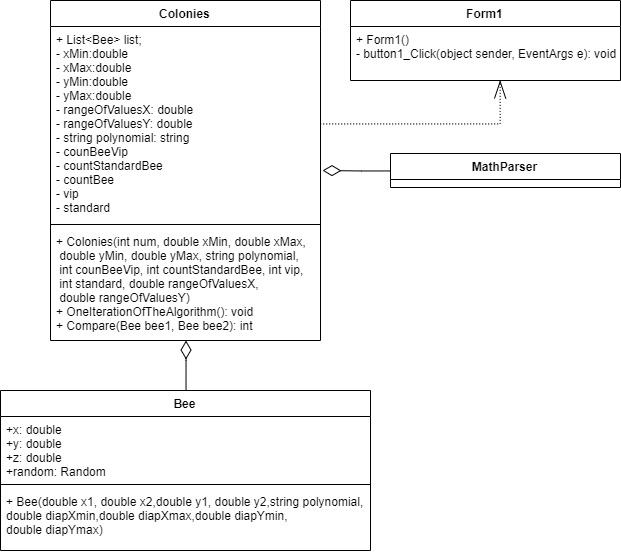


Рис. 1. UML-диаграмма

# Описание классов

Таблица 1 Описание методов класса Bee

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| Bee() | - | public | double x1, double x2,double y1, double y2,string polynomial,double diapXmin,double diapXmax,double diapYmin,double diapYmax | - | Конструктор Bee |

Таблица 2 Описание полей и свойств класса Bee

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| x | double | public | Координата Х |
| y | double | public | Координата Y |
| z | double | public | Значение функции |
| random | Random | public | Для получения рандомного числа |

Таблица 3 Описание методов класса Colonies

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| Colonies | - | public | int num, double xMin, double xMax, double yMin, double yMax, string polynomial, int counBeeVip, int countStandardBee, int vip, int standard, double rangeOfValuesX, double rangeOfValuesY | - | Конструктор класса Colonies |
| OneIterationOfTheAlgorithm() | void | public | - | - | Производит одну итерацию алгоритма |
| Compare | int | public | Bee bee1, Bee bee2 | 1,-1,0 | Форма для сравнивания двух объектов с типом данных Bee |

Таблица 4 Описание полей и свойств класса Colonies

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| list | List<Bee> | public | Хранит все точки |
| xMin | double | private | Диапоз |
| xMax | double | private | Диапоз |
| yMin | double | private | Диапоз |
| yMax | double | private | Диапоз |
| rangeOfValuesX | double | private | Окрестность поиска Х |
| rangeOfValuesY | double | private | Окрестность поиска Y |
| polynomial | string | private | Функция |
| counBeeVip | int | private | Количество дополнительных элитных точек |
| countStandardBee | int | private | Количество дополнительных стандартных точек |
| countBee | int | private | Количество точек усиленного поиска |
| vip | int | private | Количество элитных точек |
| standard | int | private | Количество стандартных точек |

Таблица 5 Описание методов класса Form1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| Form1 | - | public | - | - | Рисует панель для пользователя |
| button1\_Click | void | private | o object sender, EventArgs e | - | Запускает алгоритм и выводит минимум |

# Ход работы

Для выполнения данной лабораторной работы был выбран язык C#. Сначала пользователь должен ввести функцию в поле , после количество точек, количество элитных точек, количество точек поиска, количество дополнительных у элитных точек, количество дополнительных у обычных точек, задать диапазон по и по , окрестности поиска у и и количество итераций. После программа выведет минимум функции в диапазоне введенном пользователем.

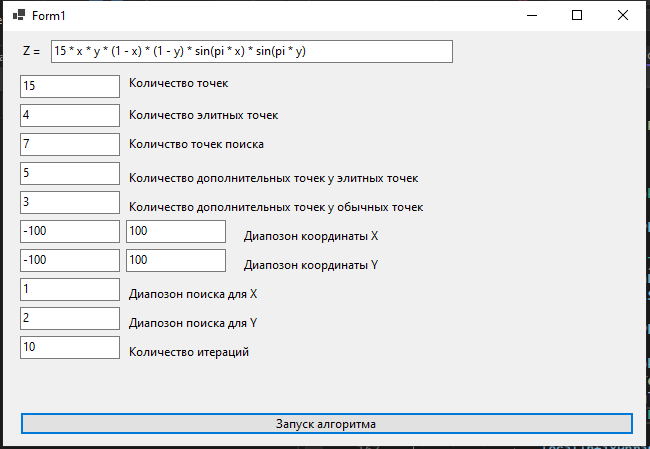
**

Рис.2.Пользовательский интерфейс

# Пример работы программы

При поиске минимума функции: , программа близка к минимальному значению, которое находится в точке (-100;-100).

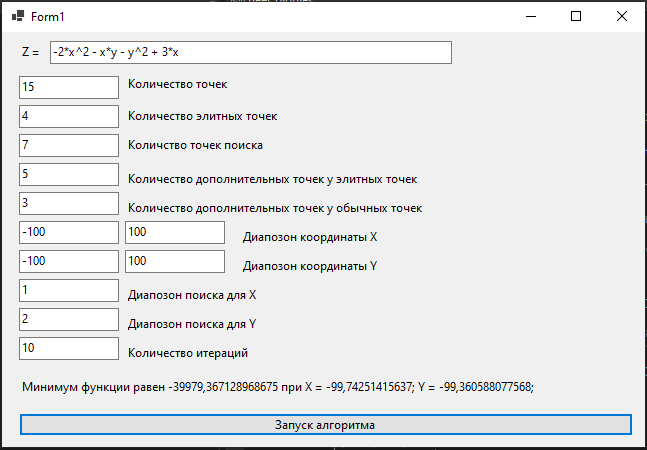


Рис.3.Поиск минимума для функции в программе

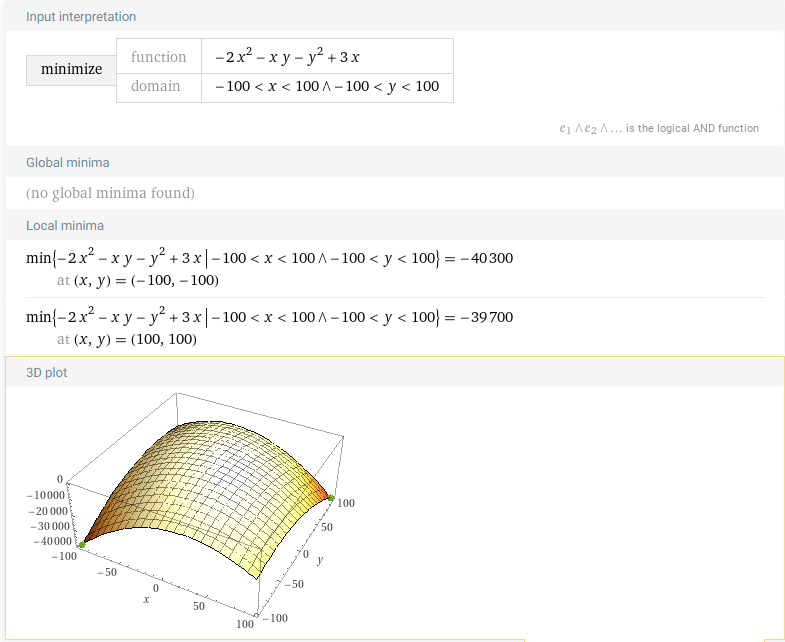


Рис.4.Поиск минимума для функции в Wolfram Alpha

При поиске минимума функции: программа близка к минимальному значению, которое находится в точке (5;6).

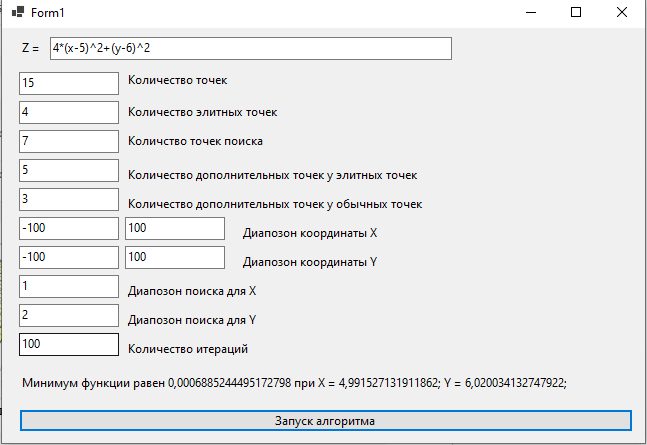


Рис.5. Поиск минимума для функции в программе

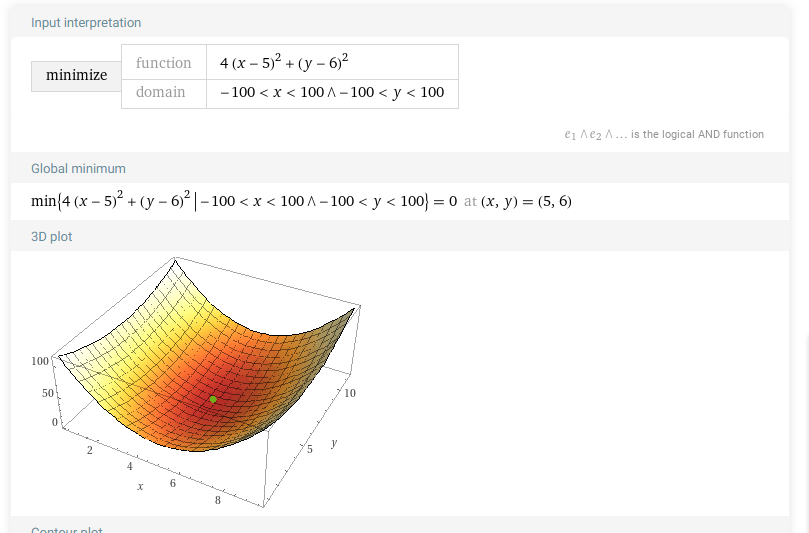


Рис.6. Поиск минимума для функции в Wolfram Alpha

При поиске минимума функции: программа близка к минимальному значению, которое находится в точке (0;-100).

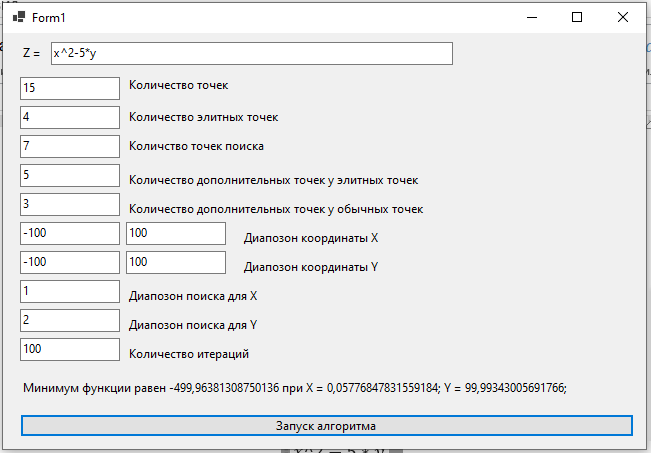


Рисунок 7. Поиск минимума для функции в программе

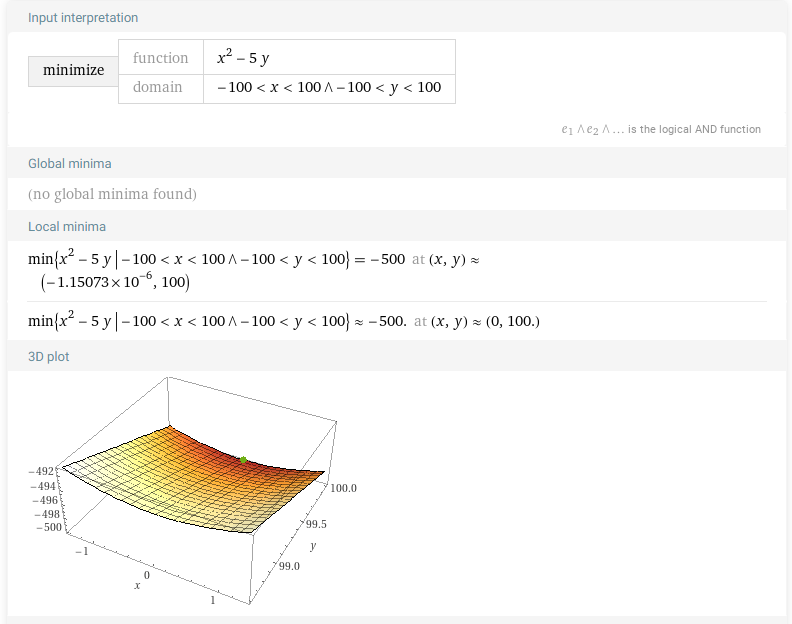


Рис.8. Поиск минимума для функции в Wolfram Alpha

# Вывод

В ходе выполнения лабораторной работы была реализован пчелиный алгоритм, который используется для определения минимума функций. Так же был реализован пользовательский интерфейс который позволяет вводить функцию и искать ее минимум.

# Список используемой литературы

1. Нахождение минимума // Вольфрам URL: <https://www.wolframalpha.com/input?i=FindMinimum> (дата обращения: 15.12.2022).
2. Алгоритм пчелиной колонии // Википедия URL: <https://ru.wikipedia.org/wiki/Алгоритм_пчелиной_колонии> (дата обращения: 15.12.2022).

# Приложение

**Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace BeeColonies

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

Colonies colonies = new Colonies(int.Parse(txb\_numberOfPoints.Text), double.Parse(txb\_minX.Text), double.Parse(txb\_maxX.Text),

double.Parse(txb\_minY.Text),double.Parse(txb\_maxY.Text),

txb\_polinom.Text, int.Parse(txb\_numberOfAdditionalPointsAtVipPoints.Text), int.Parse(txb\_numberOfAdditionalPointsAtStandardPoints.Text),

int.Parse(txb\_numberOfVipPoints.Text), int.Parse(txb\_numberOfSearchPoints.Text),

double.Parse(txb\_rangeOfValuesX.Text), double.Parse(txb\_rangeOfValuesY.Text));

for (int i = 0; i < int.Parse(txb\_numberOfIterations.Text); i++)

colonies.OneIterationOfTheAlgorithm();

label7.Text = "Минимум функции равен " + colonies.list[0].z + " при X = " + colonies.list[0].x + "; Y = " + colonies.list[0].y + ";";

label7.Visible = true;

}

private void txb\_polinom\_TextChanged(object sender, EventArgs e)

{

}

}

}

**Colonies.cs**

using Microsoft.VisualBasic;

using System;

using System.Collections.Generic;

using MathParserSpace;

namespace BeeColonies

{

class Bee

{

public double x;

public double y;

public double z;

public Random random;

public Bee(double x1, double x2,double y1, double y2,string polynomial,double diapXmin,double diapXmax,double diapYmin,double diapYmax)

{

random = new Random();

if (x1 < diapXmin)

x1 = diapXmin;

if (x2 > diapXmax)

x2 = diapXmax;

if (y1 < diapYmin)

y1 = diapYmin;

if (y2 > diapYmax)

y2 = diapYmax;

x = (x2 - x1) \* random.NextDouble() + x1;

y = (y2 - y1) \* random.NextDouble() + y1;

z = MathParser.calculate(x, y, polynomial);

}

}

class Colonies:IComparer<Bee>

{

public List<Bee> list;

private double xMin, xMax, yMin, yMax, rangeOfValuesX, rangeOfValuesY;

private string polynomial;

private int counBeeVip, countStandardBee, countBee, vip, standard;

public Colonies(int num, double xMin, double xMax, double yMin, double yMax, string polynomial, int counBeeVip, int countStandardBee, int vip, int standard, double rangeOfValuesX, double rangeOfValuesY)

{

this.xMin = xMin;

this.xMax = xMax;

this.yMin = yMin;

this.yMax = yMax;

this.polynomial = polynomial;

this.counBeeVip = counBeeVip;

this.countStandardBee = countStandardBee;

this.countBee = num - (counBeeVip + countStandardBee);

this.vip = vip;

this.rangeOfValuesX = rangeOfValuesX;

this.rangeOfValuesY = rangeOfValuesY;

this.standard = standard - vip;

list = new List<Bee>();

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

{

list.Add(new Bee(xMin, xMax, yMin, yMax, polynomial,xMin,xMax,yMin,yMax));

}

list.Sort(Compare);

list.RemoveRange(counBeeVip + countStandardBee, countBee);

}

public void OneIterationOfTheAlgorithm()// диапазон

{

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

for (int j = 0; j < counBeeVip; j++)

list.Add(new Bee((list[i].x - rangeOfValuesX), (list[i].x + rangeOfValuesX), (list[i].y - rangeOfValuesY), (list[i].y + rangeOfValuesY), polynomial, xMin, xMax, yMin, yMax));

for (int i = vip; i < vip + standard; i++)

for (int j = 0; j < countStandardBee; j++)

list.Add(new Bee((list[i].x - rangeOfValuesX), (list[i].x + rangeOfValuesX), (list[i].y - rangeOfValuesY), (list[i].y + rangeOfValuesY), polynomial, xMin, xMax, yMin, yMax));

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

list.Add(new Bee(xMin, xMax, yMin, yMax, polynomial, xMin, xMax, yMin, yMax));

list.Sort(Compare);

}

public int Compare(Bee bee1, Bee bee2)

{

if (bee1.z < bee2.z)

return -1;

if (bee1.z >= bee2.z)

return 1;

return 0;

}

}

}

**MathParser.cs**

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

//\* Designed by KiShiVi

//\*

//\* For normal work without fucking the brain,

//\* just call the static method "calculate()".

//\* In the parameters, specify x, y and the expression itself as a normal string.

//\*

//\*

//\* The following characters, operators, constants, and functions are supported:

//\* ( ) + - \* / ^ sin cos tg ctg ln pi e x y

//\*

//\* ATTENTION! There is no check for the correctness of the expression!

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MathParserSpace

{

public enum OperatorType

{

PREFIX,

OPEN\_BRACKET,

CLOSE\_BRACKET,

BINARY,

CONST,

DIGIT,

ERROR,

NULL,

PARAM,

}

internal class Token

{

public string singleToken;

public int priority;

public OperatorType type;

public Token(string in\_singleToken, int in\_priority, OperatorType in\_type)

{

this.singleToken = in\_singleToken;

this.priority = in\_priority;

this.type = in\_type;

}

public Token(Token token)

{

this.singleToken = token.singleToken;

this.priority = token.priority;

this.type = token.type;

}

}

internal class MathParser

{

private static List<Token> tokens = new List<Token>{

new Token("(", -1, OperatorType.OPEN\_BRACKET),

new Token(")", -1, OperatorType.CLOSE\_BRACKET),

new Token("+", 1, OperatorType.BINARY),

new Token("-", 1, OperatorType.BINARY),

new Token("\*", 2, OperatorType.BINARY),

new Token("/", 2, OperatorType.BINARY),

new Token("^", 3, OperatorType.BINARY),

new Token("sin", 0, OperatorType.PREFIX),

new Token("cos", 0, OperatorType.PREFIX),

new Token("tg", 0, OperatorType.PREFIX),

new Token("ctg", 0, OperatorType.PREFIX),

new Token("ln", 0, OperatorType.PREFIX),

new Token("pi", -1, OperatorType.CONST),

new Token("x", -1, OperatorType.PARAM),

new Token("y", -1, OperatorType.PARAM),

new Token("e", -1, OperatorType.CONST) };

public static double calculate(double x, double y, string infixPhrase)

{

List<Token> polishList = toPolishPhrase(infixPhrase);

Stack<double> localStack = new Stack<double>();

foreach (Token i in polishList)

{

switch (i.type)

{

case OperatorType.DIGIT:

localStack.Push(Double.Parse(i.singleToken));

break;

case OperatorType.BINARY:

switch (i.singleToken)

{

case "+": localStack.Push(localStack.Pop() + localStack.Pop()); break;

case "-": localStack.Push(-localStack.Pop() + localStack.Pop()); break;

case "\*": localStack.Push(localStack.Pop() \* localStack.Pop()); break;

case "/": localStack.Push(reverseDiv(localStack.Pop(), localStack.Pop())); break;

case "^": localStack.Push(reversePow(localStack.Pop(), localStack.Pop())); break;

}

break;

case OperatorType.PREFIX:

switch (i.singleToken)

{

case "sin": localStack.Push(Math.Sin(localStack.Pop())); break;

case "cos": localStack.Push(Math.Cos(localStack.Pop())); break;

case "tg": localStack.Push(Math.Tan(localStack.Pop())); break;

case "ctg": localStack.Push(ctg(localStack.Pop())); break;

case "ln": localStack.Push(Math.Log(localStack.Pop())); break;

case "-sin": localStack.Push(-Math.Sin(localStack.Pop())); break;

case "-cos": localStack.Push(-Math.Cos(localStack.Pop())); break;

case "-tg": localStack.Push(-Math.Tan(localStack.Pop())); break;

case "-ctg": localStack.Push(-ctg(localStack.Pop())); break;

case "-ln": localStack.Push(-Math.Log(localStack.Pop())); break;

}

break;

case OperatorType.CONST:

switch (i.singleToken)

{

case "pi": localStack.Push(3.1415926); break;

case "e": localStack.Push(2.7182818); break;

case "-pi": localStack.Push(-3.1415926); break;

case "-e": localStack.Push(-2.7182818); break;

}

break;

case OperatorType.PARAM:

switch (i.singleToken)

{

case "x": localStack.Push(x); break;

case "y": localStack.Push(y); break;

case "-x": localStack.Push(-x); break;

case "-y": localStack.Push(-y); break;

}

break;

default: throw new Exception("Something gone wrong");

}

}

return localStack.Pop();

}

private static List<Token> toPolishPhrase(string infixPhrase)

{

string localInfixPhrase = infixPhrase.Replace(" ", "");

Token prevToken;

Token token = new Token("", -2, OperatorType.NULL);

List<Token> polishList = new List<Token>();

Stack<Token> localStack = new Stack<Token>();

while (localInfixPhrase != "")

{

prevToken = token;

token = parseToken(localInfixPhrase, prevToken);

if (token.singleToken == "")

throw new Exception("program Error");

localInfixPhrase = localInfixPhrase.Substring(token.singleToken.Length);

switch (token.type)

{

case OperatorType.CONST:

switch (token.singleToken)

{

case "pi":

polishList.Add(new Token(Math.PI.ToString(), -1, OperatorType.DIGIT));

break;

case "e":

polishList.Add(new Token(Math.E.ToString(), -1, OperatorType.DIGIT));

break;

}

break;

case OperatorType.PARAM: polishList.Add(token); break;

case OperatorType.DIGIT: polishList.Add(token); break;

case OperatorType.PREFIX: localStack.Push(token); break;

case OperatorType.OPEN\_BRACKET: localStack.Push(token); break;

case OperatorType.CLOSE\_BRACKET:

if (localStack.Count == 0)

throw new Exception("Incorrect phrase");

Token tempToken = localStack.Pop();

while (tempToken.type != OperatorType.OPEN\_BRACKET)

{

polishList.Add(tempToken);

if (localStack.Count == 0)

throw new Exception("Incorrect phrase");

tempToken = localStack.Pop();

}

break;

case OperatorType.BINARY:

while (localStack.Count > 0 && (localStack.Peek().type == OperatorType.PREFIX ||

localStack.Peek().priority >= token.priority))

polishList.Add(localStack.Pop());

localStack.Push(token);

break;

default: throw new Exception("Something gone wrong"); break;

}

}

while (localStack.Count > 0)

{

if (localStack.Peek().type == OperatorType.OPEN\_BRACKET ||

localStack.Peek().type == OperatorType.CLOSE\_BRACKET)

throw new Exception("Incorrect phrase (brackets)");

polishList.Add(localStack.Pop());

}

return polishList;

}

private static Token parseToken(string infixPhrase, Token prevToken)

{

if (infixPhrase == "")

throw new Exception("parseToken error");

string outToken = "";

int iCounter = 0;

if (Char.IsDigit(infixPhrase[0]))

{

while (iCounter < infixPhrase.Length && Char.IsDigit(infixPhrase[iCounter]))

{

outToken += infixPhrase[iCounter];

++iCounter;

}

return new Token(outToken, -1, OperatorType.DIGIT);

}

foreach (Token token in tokens)

{

//Сразу отсеиваем токены, которые длиннее мат. выражения

if (infixPhrase.Length < token.singleToken.Length)

continue;

if (infixPhrase.Substring(0, token.singleToken.Length) == token.singleToken)

{

// check for unary + and -

if (((token.singleToken == "+") || (token.singleToken == "-")) &&

((prevToken.type == OperatorType.NULL) ||

(prevToken.type == OperatorType.BINARY) ||

(prevToken.type == OperatorType.OPEN\_BRACKET)))

{

outToken += token.singleToken;

Token nextToken = parseToken(infixPhrase.Substring(token.singleToken.Length), token);

outToken += nextToken.singleToken;

return new Token(outToken, -1, nextToken.type);

}

else

return new Token(token);

}

}

// not parsed

return new Token("", -2, OperatorType.ERROR);

}

private static double reversePow(double a, double b)

{

return Math.Pow(b, a);

}

private static double reverseDiv(double a, double b)

{

return b / a;

}

private static double ctg(double a)

{

return Math.Cos(a) / Math.Sin(a);

}

}

}

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace BeeColonies

{

internal static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.SetHighDpiMode(HighDpiMode.SystemAware);

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}