**Московский Авиационный Институт**

(Национальный исследовательский университет)

**Факультет прикладной математики и информационных технологий**

**Курсовая работа по теме**

**«Инструментальные средства разработки прикладных программных систем».**

Выполнил студент группы: М8О-205Б-19

Васильев А. В.

Руководитель: Семенов А. В.

Оценка:

Дата:

Москва, 2020

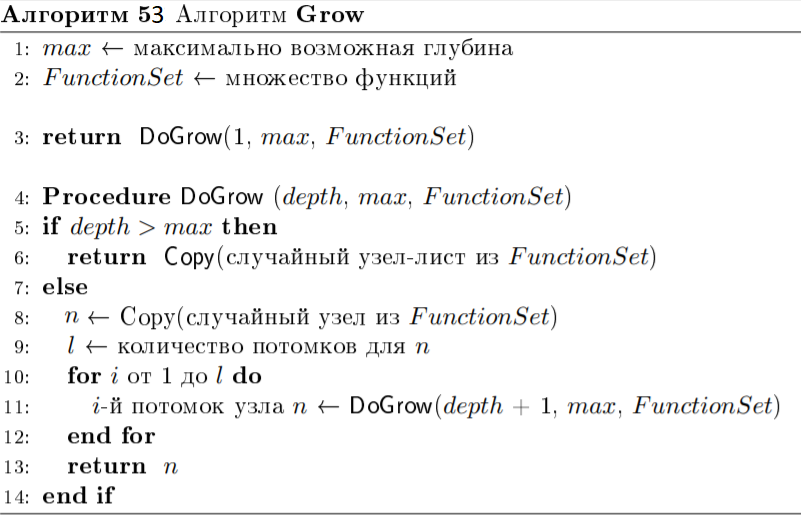
**Алгоритм №53:** «The Grow Algorithm».

**Постановка задачи.**

Алгоритм генерирует случайные деревья из некоторых функций в глубину до заданного значения (глубина дерева не обязательно равна максимально заданной).

Набор функций, из которых генерируется дерево, в моей программе изначально ограничен 18 различными переменными (листами) и простыми функциями, но в программе предусмотрена возможность расширения списка составляющих любыми функциями с количеством аргументов от 0 (лист) до 5. При генерации дерева происходит его отрисовка и вывод получившейся функции, в которую дерево «складывается».

**Алгоритм на псевдокоде и его описание.**



На вход поступает список функций, из которых генерируется дерево, а также максимальная глубина, больше которой дереву “расти” не положено. Сам алгоритм DoGrow является рекурсивным: начинается построение дерева с вызова процедуры DoGrow с 1 в качестве аргумента текущей глубины – depth. С каждым новым вызовом аргумент depth будет расти на 1, тем самым не позволяя алгоритму превысить максимальную глубину.

Внутри процедуры DoGrow сначала происходит проверка на то, не достигло ли дерево своей максимальной глубины, и если нет, то алгоритм выбирает случайную новую функцию из заданного списка FunctionSet и строит «ветки» дальше – для каждого аргумента выбранной функции, если аргументы у таковой имеются.

Если же дерево достигло максимальной глубины, алгоритм выбирает из FunctionSet и возвращает в качестве функции один случайный лист – переменную (или константу) без аргументов.

На выходе имеем случайно сгенерированное дерево из заданных функций.

**Исходный код программы.**

Программа является приложением на WinForms и использует архитектурный паттерн MVC. Вам стоит ознакомиться с кодом через Visual Studio – для удобства, чтобы все ссылки кода читались и были понятны. В программе реализован не только сам алгоритм, но и многое другое. Из приёмов и материалов пройденного курса реализованы паттерн Singleton, MVC, модальные окна, рисование на pictureBox.

Далее привожу некоторые куски кода, вырванные из контекста:

public struct VD\_Type\_Func

{

public int Offspring { get; set; }

public string SomeFunction { get; set; }

public VD\_Type\_Func[] child { get; set; }

public bool Between { get; set; }

public VD\_Type\_Func(string SomeFunction, int Offspring, bool Between)

{

this.Offspring = Offspring;

this.SomeFunction = SomeFunction;

this.child = new VD\_Type\_Func[Offspring];

this.Between = Between;

}

public VD\_Type\_Func(string SomeFunction, int Offspring)

{

this.Offspring = Offspring;

this.SomeFunction = SomeFunction;

this.child = new VD\_Type\_Func[Offspring];

this.Between = false;

}

public VD\_Type\_Func(string SomeFunction, int Offspring, bool Between, VD\_Type\_Func[] children)

{

this.Offspring = Offspring;

this.SomeFunction = SomeFunction;

this.child = children;

this.Between = Between;

}

public VD\_Type\_Func(bool Between, int Offspring, string SomeFunction) : this(SomeFunction, Offspring, Between) { }

}

public class GrowAlgorithm : IParser<GenSettings, VD\_Type\_Func>

{

List<VD\_Type\_Func> ListOfFunc;

VD\_Type\_Func FinalTree;

Random random1, random2;

int ZeroArgs;

public GrowAlgorithm()

{

FinalTree = new VD\_Type\_Func();

random1 = new Random();

random2 = new Random();

ZeroArgs = 5;

ListOfFunc = new List<VD\_Type\_Func>

{

new VD\_Type\_Func ( false, 0, "x"),

new VD\_Type\_Func ( false, 0, "y"),

new VD\_Type\_Func ( false, 0, "z"),

new VD\_Type\_Func ( false, 0, "1"),

new VD\_Type\_Func ( false, 0, "k"),

new VD\_Type\_Func ( true, 2, "+"),

new VD\_Type\_Func ( true, 2, "-"),

new VD\_Type\_Func ( true, 2, "\*"),

new VD\_Type\_Func ( true, 2, "/"),

new VD\_Type\_Func ( true, 2, "%"),

new VD\_Type\_Func ( true, 2, "\\"),

new VD\_Type\_Func ( false, 1, "sin"),

new VD\_Type\_Func ( false, 1, "cos"),

new VD\_Type\_Func ( false, 1, "tg"),

new VD\_Type\_Func ( false, 1, "ctg"),

new VD\_Type\_Func ( false, 1, "sqrt"),

new VD\_Type\_Func ( false, 1, "abs"),

new VD\_Type\_Func ( true, 2, "^" )

};

SortFunc(ListOfFunc);

}

public VD\_Type\_Func GetFinalTree()

{

return FinalTree;

}

public void AddOneMoreFunc(VD\_Type\_Func onemore)

{

ListOfFunc.Add(onemore);

if(onemore.Offspring == 0)

{

ZeroArgs++;

}

SortFunc(ListOfFunc);

}

private class VD\_Type\_Func\_Comparer : IComparer<VD\_Type\_Func>

{

public int Compare(VD\_Type\_Func cl1, VD\_Type\_Func cl2)

{

return cl1.Offspring == cl2.Offspring ? 0 :

(cl1.Offspring < cl2.Offspring ? -1 : 1);

}

}

static List<VD\_Type\_Func> SortFunc(List<VD\_Type\_Func> lf)

{

lf.Sort(new VD\_Type\_Func\_Comparer());

return lf;

}

static int CountLeafInSortedFunc(List<VD\_Type\_Func> lf)

{

int k = 0;

bool flag = true;

int kmax = lf.Count();

while (flag && k < kmax)

{

if (lf[k].Offspring == 0) { k++; } else { flag = false; }

}

return k;

}

public VD\_Type\_Func DoGrow(int max, List<VD\_Type\_Func> FunctionSet)

{

return DoGrow(1, max, FunctionSet);

}

public VD\_Type\_Func DoGrow(int max)

{

return DoGrow(1, max, ListOfFunc);

}

public void VoidDoGrow(int max)

{

FinalTree = new VD\_Type\_Func();

FinalTree = DoGrow(max);

CreatePic();

}

public void CreatePic()

{

foreach (IObserver ls in listeners)

{

ls.UpdateForm();

}

}

public bool IsTreeNotNull()

{

return FinalTree.SomeFunction != null;

}

public string FullFunction()

{

return FullFunction(FinalTree);

}

string FullFunction(VD\_Type\_Func finaltree)

{

string s = "";

if (finaltree.Offspring == 0) { return finaltree.SomeFunction; }

else

{

if (finaltree.Between == true)

{

s += "(";

for (int i = 0; i < finaltree.Offspring - 1; i++)

{

s += FullFunction(finaltree.child[i]) + finaltree.SomeFunction;

}

s += FullFunction(finaltree.child[finaltree.Offspring - 1]) + ")";

}

else

{

s += finaltree.SomeFunction + "(";

for (int i = 0; i < finaltree.Offspring - 1; i++)

{

s += FullFunction(finaltree.child[i]) + ", ";

}

s += FullFunction(finaltree.child[finaltree.Offspring - 1]);

s += ")";

}

}

return s;

}

private VD\_Type\_Func DoGrow(int depth, int max, List<VD\_Type\_Func> FunctionSet)

{

if (depth >= max)

{

VD\_Type\_Func vd = FunctionSet[random1.Next(ZeroArgs)];

return vd;

}

else

{

int d = depth + 1;

VD\_Type\_Func n = FunctionSet[random2.Next(FunctionSet.Count)];

VD\_Type\_Func newfuncstruct = new VD\_Type\_Func(n.SomeFunction, n.Offspring, n.Between);

int l = n.child.Length;

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

{

VD\_Type\_Func bs = DoGrow(d, max, FunctionSet);

VD\_Type\_Func newfunc = new VD\_Type\_Func(bs.SomeFunction, bs.Offspring, bs.Between, bs.child);

newfuncstruct.child[i] = newfunc;

}

return newfuncstruct;

}

}

private ArrayList listeners = new ArrayList();

public void Register(IObserver o)

{

listeners.Add(o);

o.UpdateForm();

}

public void Deregister(IObserver o)

{

listeners.Remove(o);

}

public void UpdateObservers()

{

foreach (IObserver ob in listeners)

{

ob.UpdateForm();

}

}

public VD\_Type\_Func Parse(GenSettings T)

{

return DoGrow(T.Max, T.FunctionSet);

}

}

public interface IObserver

{

void UpdateForm();

void SomeErrorMessage();

void BigNumbersError();

}

public interface IGetData<T>

{

T GetData();

}

public interface IParser<TIn, TOut>

{

TOut Parse(TIn T);

}

**Отрисовка.**

Отрисовывается дерево на pictureBox-е на главной форме, вот функции, отвечающие за это. ContinueCreation также является рекурсивным.

private int maxvershin;

public void CreatePicture(object sender, PaintEventArgs e, int wed, int hei)

{

Brush brush1 = new SolidBrush(Color.Crimson);

Color color1 = Color.Black;

Font font1 = new Font(FontFamily.GenericSansSerif, 9);

VD\_Type\_Func tree = model.GetFinalTree();

int mid = wed / 2;

if(maxvershin < 1 && maxvershin > 5)

{

maxvershin = 5;

}

int dist = hei/(maxvershin + 1);

e.Graphics.DrawString(tree.SomeFunction, font1, brush1, mid - 5, dist / 2);

if (tree.Offspring != 0)

{

mid = wed / (tree.Offspring + 1);

for (int i = 0; i < tree.Offspring; i++)

{

ContinueCreation(sender, e, mid \* (i + 1), dist, wed/2 , dist/2 + 12 , tree.child[i], mid, mid \* i, 2);

}

}

}

private void ContinueCreation(object sender, PaintEventArgs e, int wed, int dist, int xline, int yline, VD\_Type\_Func newtree, int nowrazdel, int bigrazdel, int level)

{

Brush brush1 = new SolidBrush(Color.Crimson);

Brush brush2 = new SolidBrush(Color.Black);

Font font1 = new Font(FontFamily.GenericSansSerif, 9);

Pen pen1 = new Pen(brush2);

e.Graphics.DrawLine(pen1, xline, yline, wed, dist \* (level - 1) - 5);

e.Graphics.DrawString(newtree.SomeFunction, font1, brush1, wed - 5, dist \* (level - 1));

if (newtree.Offspring != 0)

{

int smallrazdel = nowrazdel / (newtree.Offspring );

for (int i = 0; i < newtree.Offspring; i++)

{

int newbigrazdel = bigrazdel + (1/2)\*nowrazdel + (smallrazdel \* i);

ContinueCreation(sender, e, bigrazdel + (smallrazdel \* (i + 1)) , dist, wed , dist \* (level - 1) + 12, newtree.child[i],smallrazdel, newbigrazdel , level + 1);

}

}

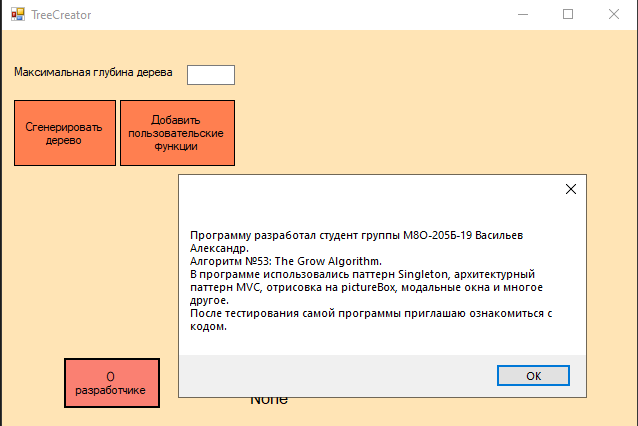
}

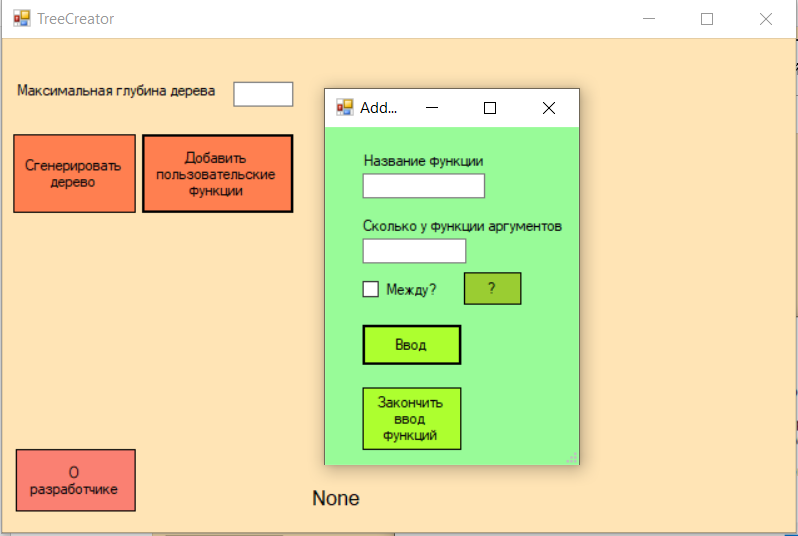
**Демонстрация функциональных возможностей программы.**

При запуске программы пользователя встречает главная форма.



По кнопке «О разработчике» можно вывести на экран MessageBox с информацией о разработчике.

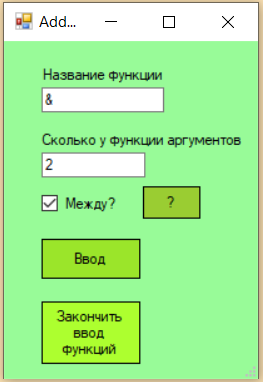
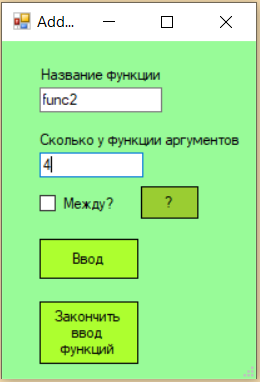


По кнопке «Добавить пользовательские функции» открывается модальная форма.

По кнопке «?» выводится MessageBox с краткой инструкцией по использованию окна добавления новых функций.

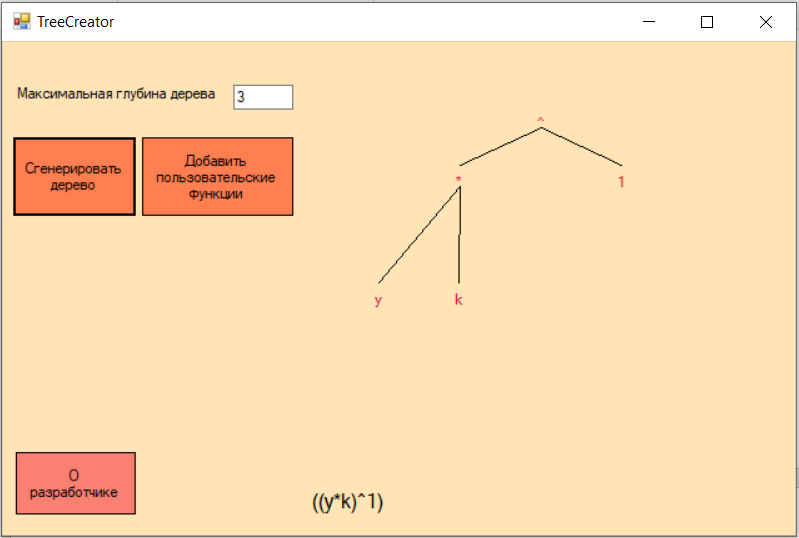
Давайте добавим пару новых функций. Например, первой будет «&», у этой функции 2 аргумента, и она ставится между аргументов.

Второй будет «func2», у этой функции будет 4 аргумента, и она будет записываться как func2(x1, x2, x3, x4).

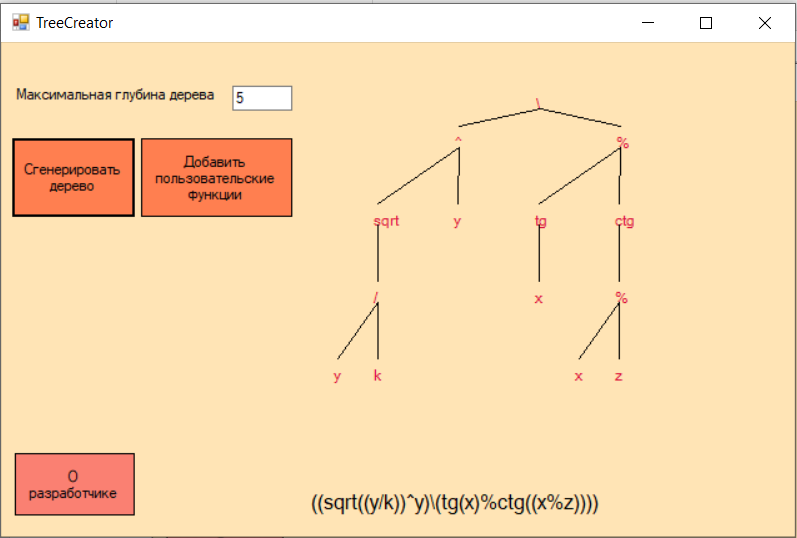
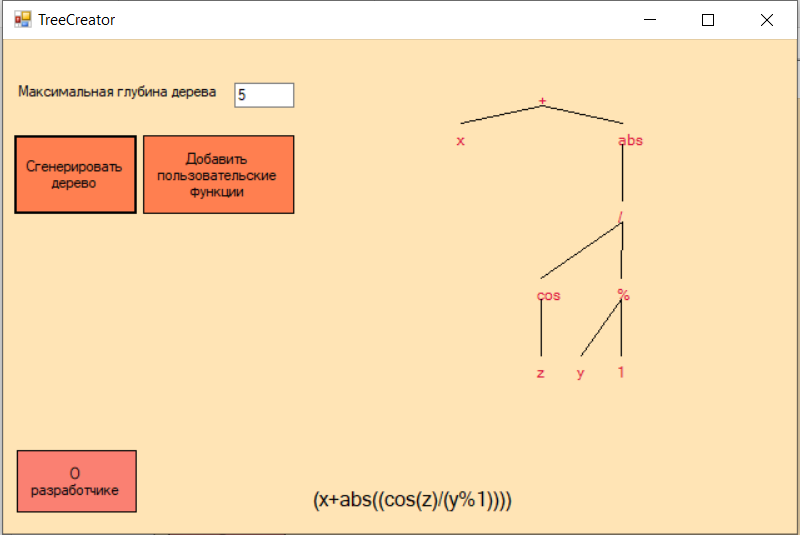
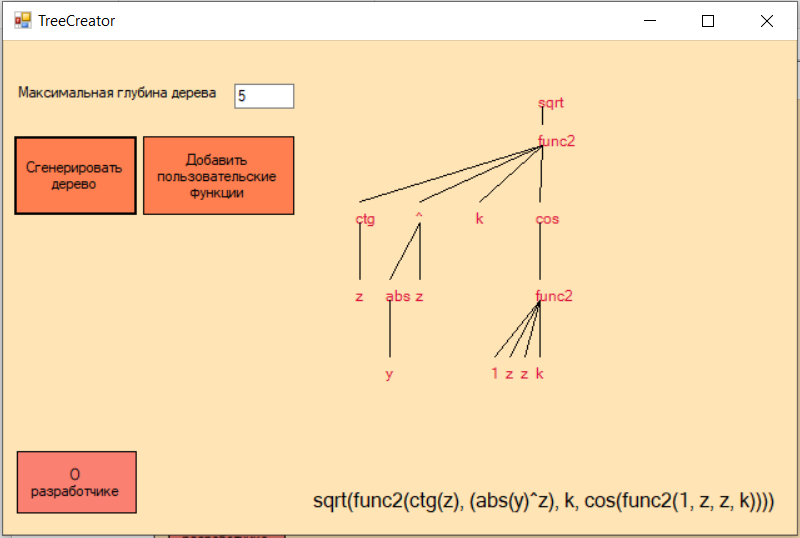
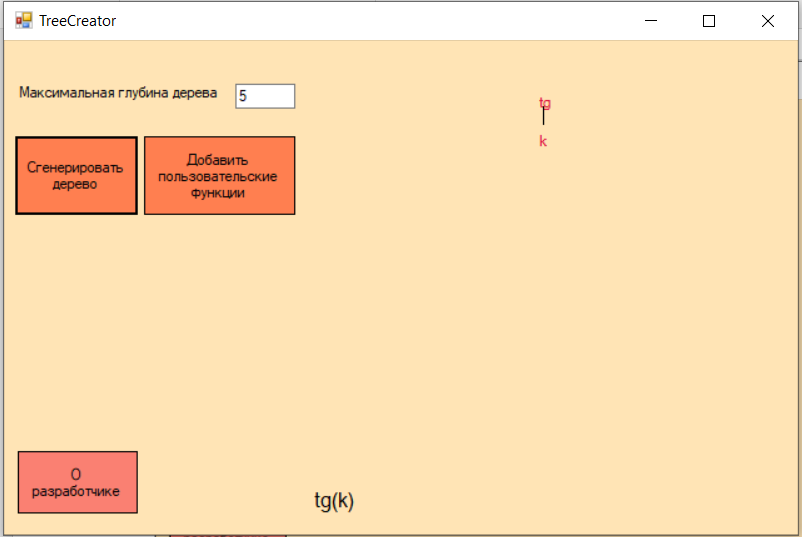
Заканчиваем ввод пользовательских функций и возвращаемся на главную форму.

Вводим желаемую максимальную глубину дерева (для первого раза пусть будет 3), нажимаем на кнопку «Сгенерировать дерево».



Программа генерирует и отрисовывает дерево, а также подписывает, как выглядит функция, заданная деревом.

Далее привожу несколько скриншотов с примерами работы программы.



**Используемое программное и аппаратное обеспечение.**

Microsoft Visual Studio 2019, .net framework 4.8,

Windows 10.

**Вывод.**

При выполнении данной работы я закрепил изученные на курсах ООП и ИСРППС навыки, лучше изучил структуру MVC и рисование, а также в целом получил дополнительный опыт программирования на языке C#, платформе .Net.