Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования

Разработка приложения под Windows «Калькулятор»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ
по дисциплине «Операционные системы»
ЮУрГУ–010302.2021.152.ПЗ КР

Автор работы,
студент группы ЕТ-312
_____ / Д.П. Чумаков
«____» _____ 2021 г.

Руководитель работы,
доцент кафедры ПМиП
_____ / Е.Ю. Алексеева
«____» _____ 2021 г.

Работа защищена с оценкой
_____
«____» _____ 2021 г.

Челябинск 2021

АННОТАЦИЯ

Чумаков Д.П. Разработка приложения под Windows «Калькулятор».– Челябинск: ЮУрГУ, ЕТ-312, 34 с., 16 ил., библиогр. список – 3 наим, 2 табл.

Целью работы является разработка приложения под Windows калькулятора с использованием среды для визуального программирования Microsoft Visual Studio 2019 Windows Forms.

В первом разделе приводится постановка задачи. Во втором разделе описывается математическая модель. Третий раздел включает в себя описание алгоритма решения данной задачи. В приложении приведен исходный код программы.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: разработать приложение под Windows калькулятора в среде визуального программирования Microsoft Visual Studio 2019 Windows Forms.

Задачи работы:

– привести постановку задачи;

– провести анализ предметной области;

– привести схему алгоритма решения;

– определить, какие визуальные компоненты будут использованы;

– привести инструкцию по применению данной программы.

1 ПОСТАНОВКА ЗАДАЧИ

Необходимо разработать компьютерную программу, реализующую простой калькулятор, на языке программирования C#. Среда разработки – Microsoft Visual Studio 2019 Windows Forms (.Net Framework) 4.7.2.

Данная программа реализует калькулятор, вычисляющий арифметическое выражение со скобками, включающее в себя операции сложения, вычитания, умножения, деления, унарного минуса, возведения в степень, квадратного корня. Например, 2+2*2.

Главное окно содержит поле ввода для выражения, поле отображения результата, кнопки для редактирования выражения, меню справки и «О программе».

Управление калькулятором производится как с помощью мыши, так и с помощью клавиатуры.

После запуска приложения на экране появляется главное окно (рисунок 1).

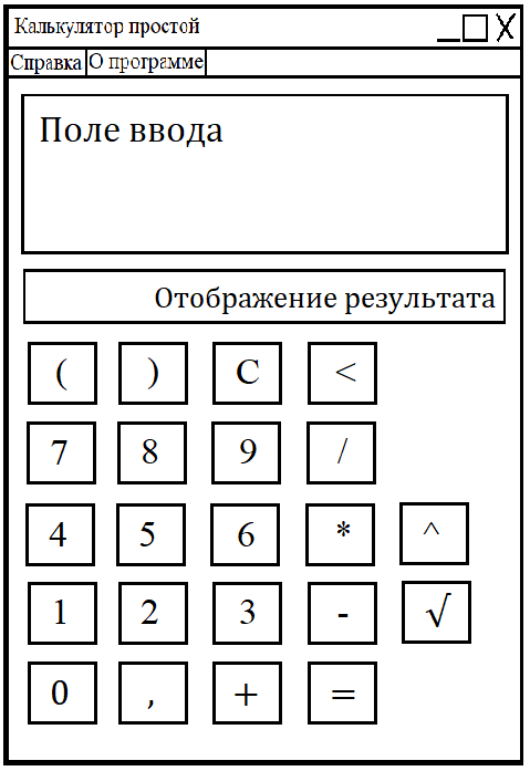

Рисунок 1.1 – Главное окно

Для ввода выражения нужно либо щелкнуть по соответствующей кнопке с цифрой или операцией, либо нажать клавишу на клавиатуре, после чего ввод отобразится в верхнем поле, а в нижнем тут же будет отображен результат (рисунок 2).

Для получения завершения вычисления нужно либо нажать кнопку «=», либо нажать клавишу «Enter» на клавиатуре, главное окно примет вид, как на рисунке 3.

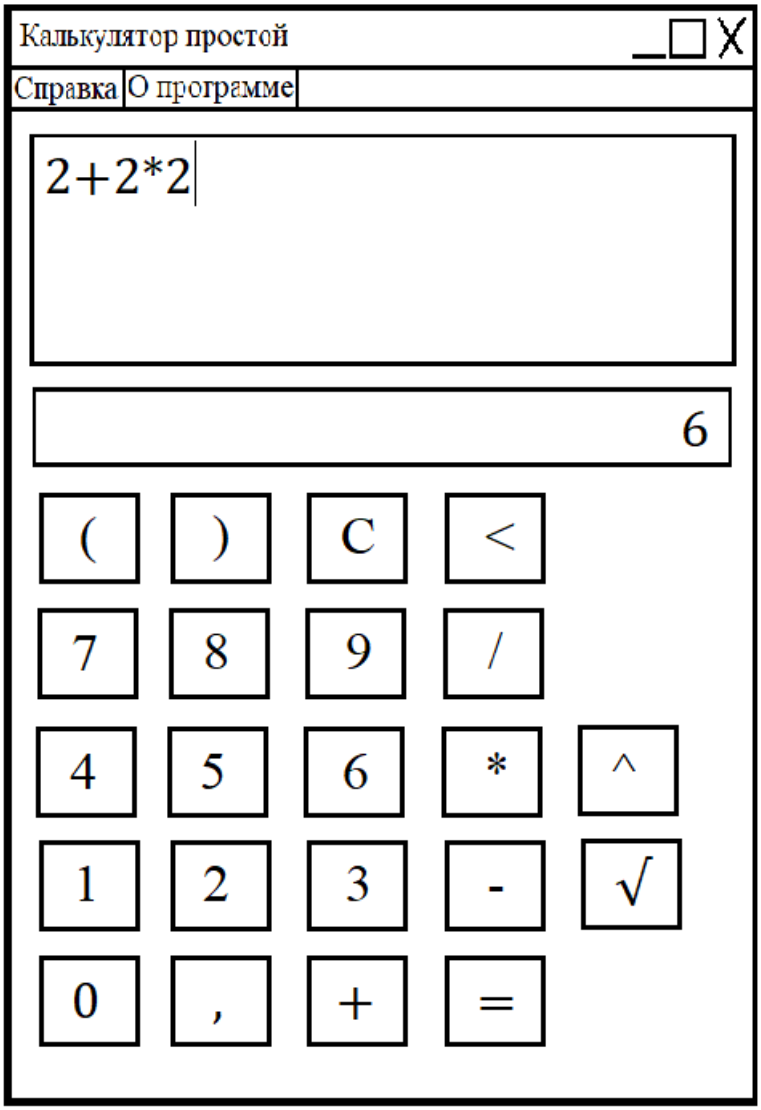

Рисунок 1.2 – Пример работы

Если введенное выражение некорректно (не согласованы скобки, деление на ноль, квадратный корень из отрицательного числа и т.д.), то результат не отобразится (рисунок 4).

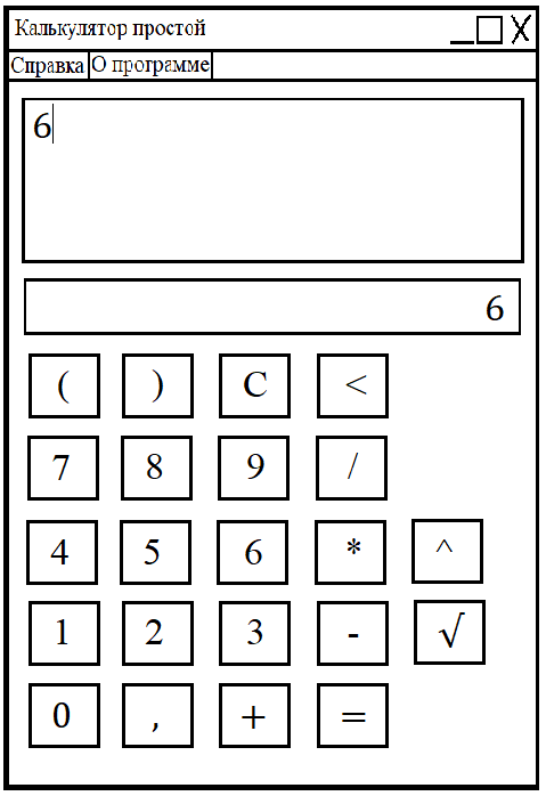

Рисунок 1.3 – Завершение работы

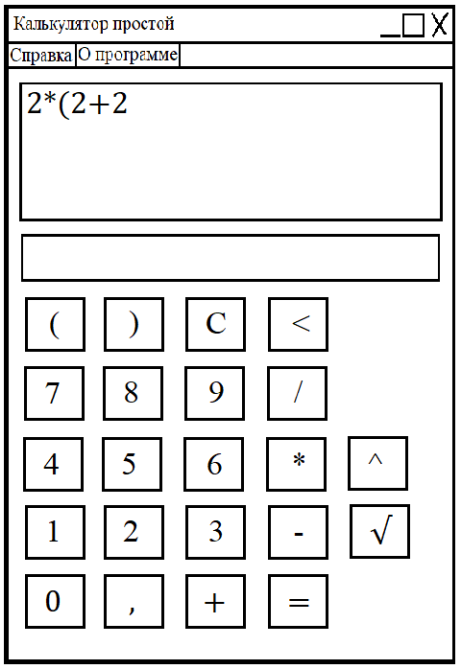

Рисунок 1.4 – Некорректное выражение

Если же пользователь нажмет кнопку «=» или «Enter», то выведется соответствующее сообщение об ошибке (рисунок 5).

При нажатии кнопки «Справка» выводится окно с руководством пользования (рисунок 6).

При нажатии кнопки «О программе» выводится окно с информацией о программе и ее авторе (рисунок 7).

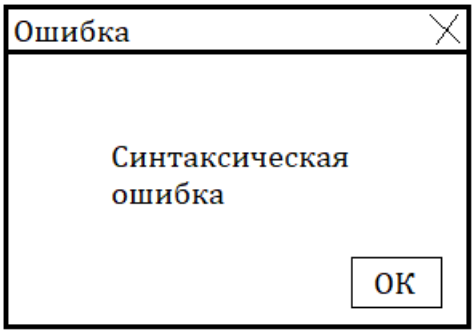

Рисунок 1.5 – Сообщение об ошибке

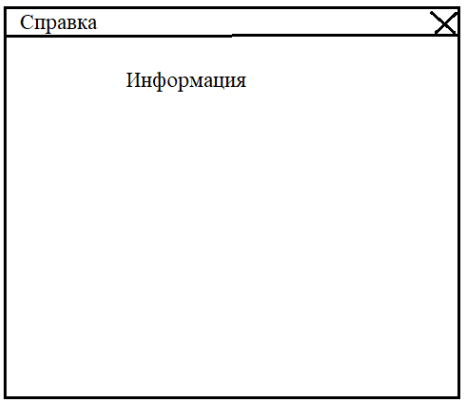

Рисунок 1.6 – Справка

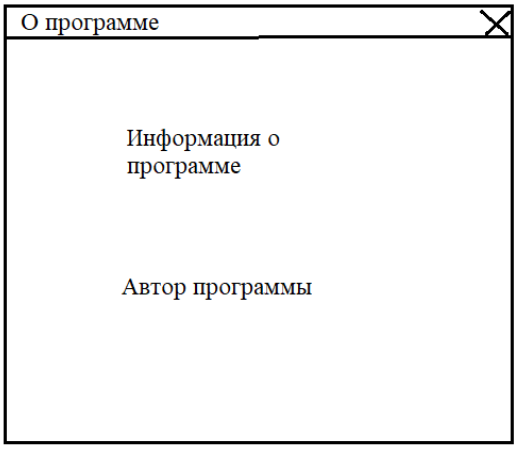

Рисунок 1.7 – О программе

2 МАТЕМАТИЧЕСКАЯ МОДЕЛЬ

Для вычисления арифметического выражения со скобками применяется обратная польская запись [1].

Приведем алгоритм преобразования из инфиксной нотации в обратную польскую. Здесь имеются входная, содержащая исходное выражение в инфиксной форме, и выходная строка, в которой после работы алгоритма содержится выражение в обратной польской нотации. Токен – это число или операция. В ходе вычисления используется стек операций.

Алгоритм:

1. Пока есть ещё токены для чтения во входной строке:

   1.1 Читаем очередной токен.

   1.2 Если токен является числом, то добавляем его к выходной строке.

   1.3 Если токен является унарной префиксной функцией (например, sin — синус) или открывающей скобкой, помещаем его в стек.

   1.4 Если токен является закрывающей скобкой:

До тех пор, пока верхним элементом стека не станет открывающая скобка, выталкиваем элементы из стека в выходную строку. При этом открывающая скобка удаляется из стека, но в выходную строку не добавляется.

Если стек закончился раньше, чем мы встретили открывающую скобку, это означает, что в выражении либо неверно поставлен разделитель, либо не согласованы скобки.

1.5 Если токен является бинарной операцией «op», тогда:

1) пока на вершине стека унарная префиксная функция

ИЛИ операция на вершине стека приоритетнее «op»

ИЛИ операция на вершине стека левоассоциативная с приоритетом как у «op», то

выталкиваем верхний элемент стека в выходную строку;

2) помещаем операцию «op» в стек.

2. Когда входная строка закончилась, выталкиваем все токены из стека в выходную строку. В стеке должны были остаться только символы операций; если это не так, значит в выражении не согласованы скобки.

Пример работы алгоритма:

Вход: 2*(3+4)

Таблица 2.1 – Преобразование выражения

№	Токен	Действие	Стек	Выход
0	∅	∅	∅	∅
1	2	Добавим 2 к выходу. Шаг 1.2	∅	2
2	*	Добавим * в стек. Шаг 1.5	*	2
3	(	Добавим (в стек. Шаг 1.3	*(	2
4	3	Добавим 3 к выходу. Шаг 1.2	*(	2 3
5	+	Добавим + в стек. Шаг 1.5	*(+	2 3
6	4	Добавим 4 к выходу. Шаг 1.2	*(+	2 3 4
7	)	Выталкиваем из стека + (. Шаг 1.4	*	2 3 4 +
8	∅	Выталкиваем из стека *. Шаг 2	∅	2 3 4 + *

Алгоритм вычисления выражения в обратной польской нотации.

1. Обработка входного токена
   1.1 Если на вход подан операнд, он помещается на вершину стека.
   1.2 Если на вход подан знак операции, то соответствующая операция выполняется над требуемым количеством значений, извлечённых из стека, взятых в порядке добавления. Результат выполненной операции кладётся на вершину стека.
2. Если еще есть токены для чтения, перейти к шагу 1.
3. После полного прочтения токенов результат вычисления лежит на вершине стеке.

Пример вычисления:

Вход 2 3 4 + *

Таблица 2.2 – Вычисление выражения

№	Токен	Действие	Стек
0	∅	∅	∅
1	2	Положим 2 в стек. Шаг 1.1	2
2	3	Положим 3 в стек. Шаг 1.1	2 3
3	4	Положим 4 в стек. Шаг 1.1	2 3 4
4	+	Достанем 3 и 4. Положим 3+4=7. Шаг 1.2	2 7
5	*	Достанем 2 и 7. Положим 2*7=14. Шаг 1.2	14
6	∅	Результат 14. Шаг 3.	14

3 ОПИСАНИЕ АЛГОРИТМА РЕШЕНИЯ

3.1 Используемые классы

Для вычисления выражения был разработан статический класс, интерфейс которого приведен ниже (файл Calculation.cs).

```
public static class Calculation
{
// виды ошибок
public enum TypesErros { NoError, Lexical, Syntax, DivideByZero,
RootNegative };

// единственный public метод, на вход подается выражение в виде
// строки, возвращает число в виде строки
        public static string Calculate(string expression, out
TypesErros error)
        {...}
}
```

3.2 Схема алгоритма работы программы

Основной алгоритм

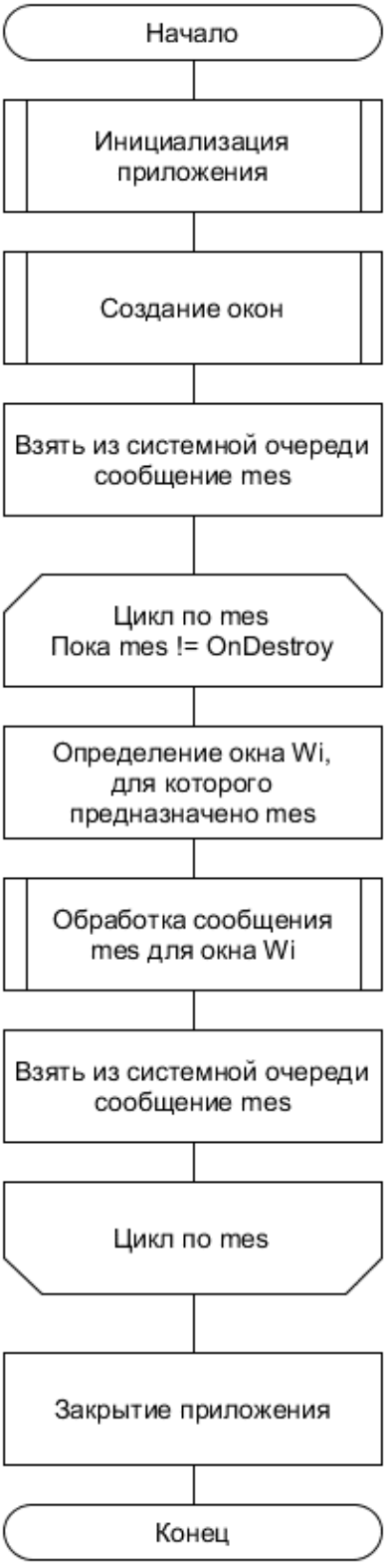

Рисунок 3.1 – Схема основного алгоритма

Вспомогательный алгоритм «Обработка сообщения mes»

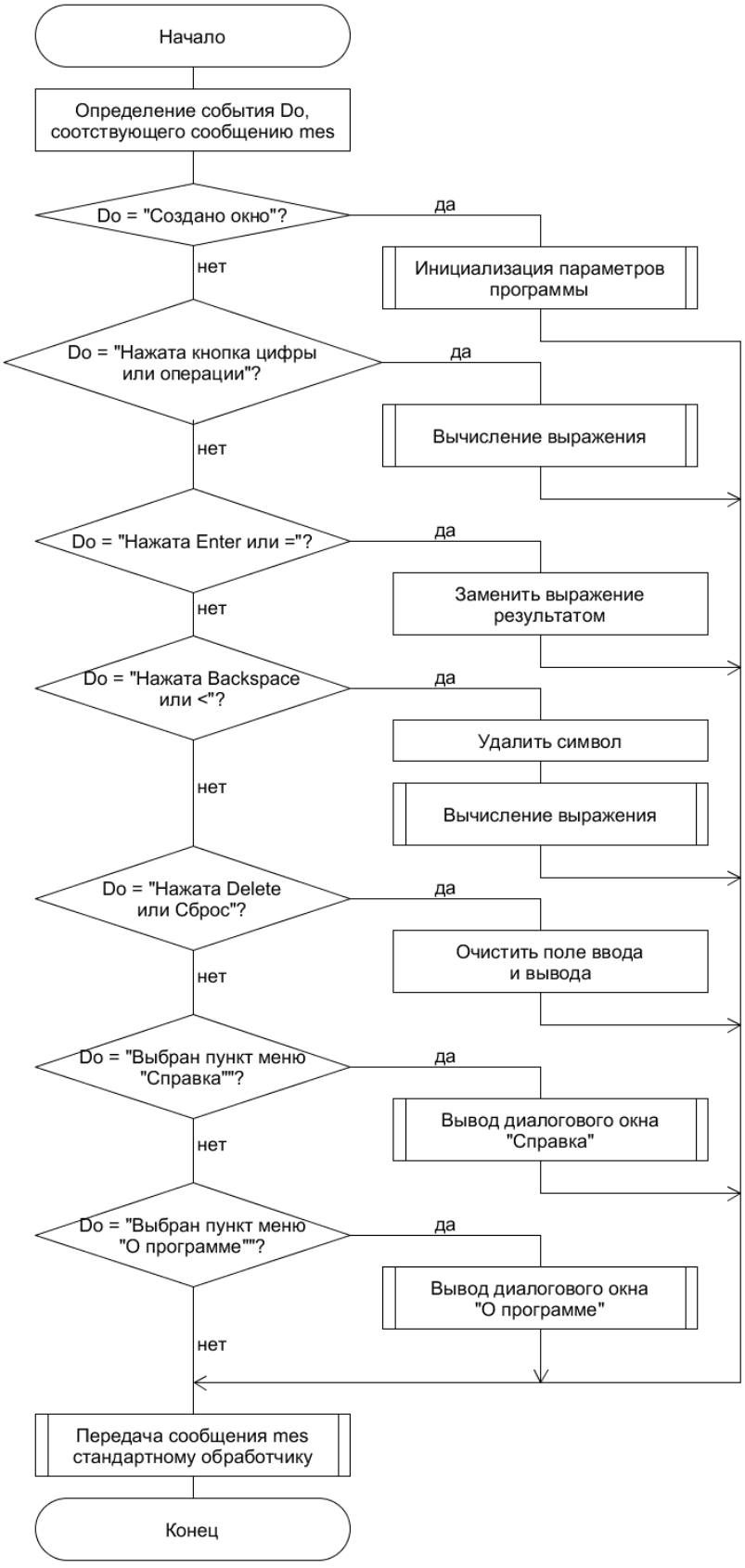

Рисунок 3.2 – Схема вспомогательного алгоритма «Обработка сообщения mes»

Вспомогательный алгоритм «Вычисление выражения»

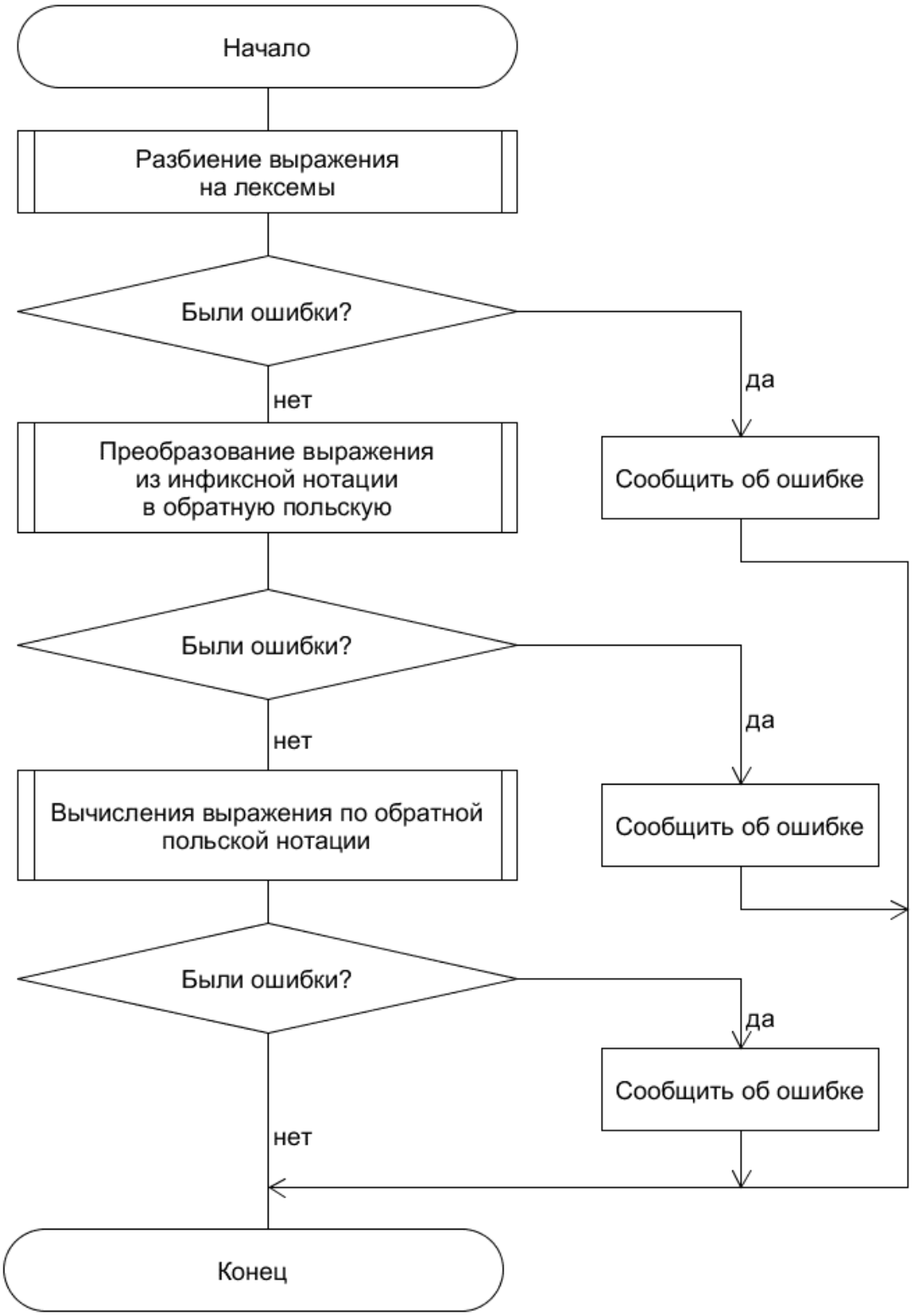

Рисунок 3.3 – Схема вспомогательного алгоритма «Вычисление выражения»

3.3 Разработка пользовательского интерфейса

На рисунке 3.4 изображен дизайн главной формы (Form1). В ней расположено главное меню (menuStrip1), поля для ввода выражения, отображения результата inputFieldTextBox, outputFieldTextBox соответственно. Ниже находятся кнопки для управления калькулятором с помощью мыши (тип Button).

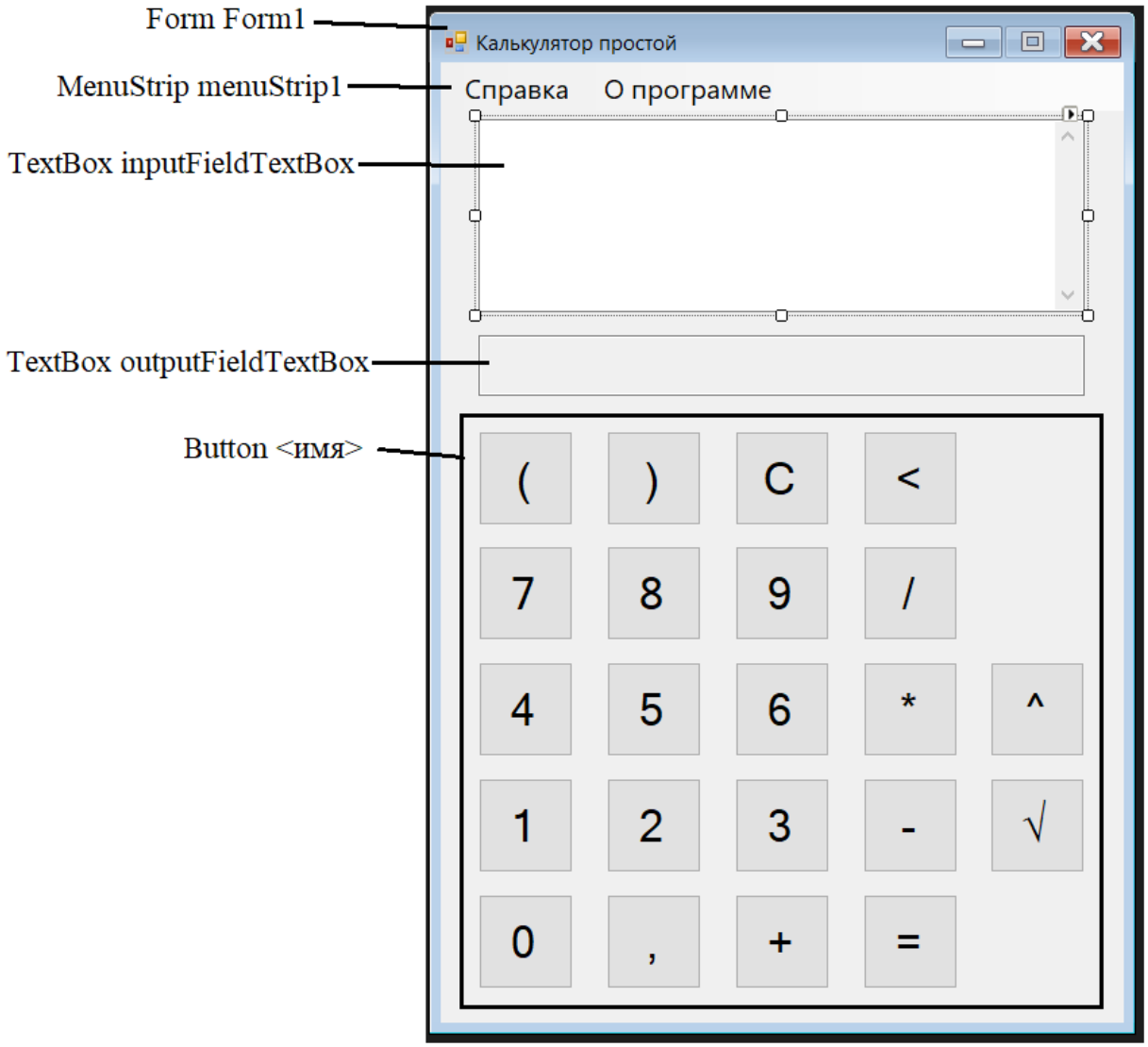

Рисунок 3.4 – Форма

3.4 Инструкция по применению разработанной программы

После запуска программа готова к вводу выражения как с помощью кнопок, так и с помощью клавиатуры. Результат пересчитывается автоматически после каждого изменения в поле ввода (рисунок 3.5). Выражение в поле ввода можно редактировать с помощью каретки (рисунок 3.6).

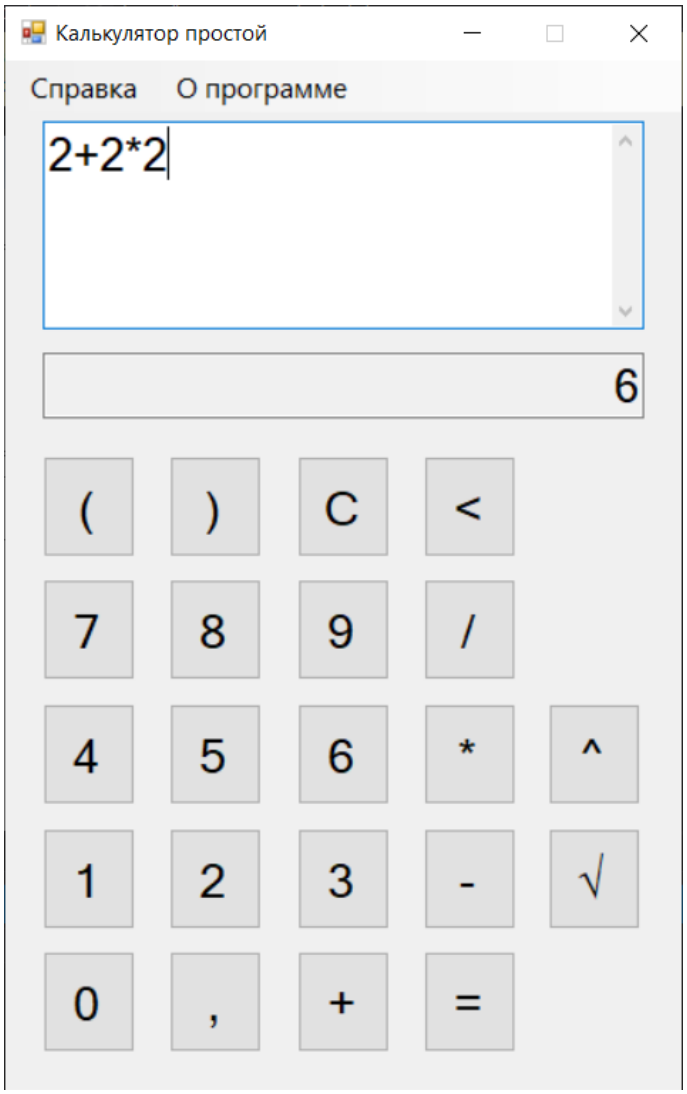

Рисунок 3.5 – Ввод выражения

Если выражение будет некорректным, поле отображения результата будет пустым (рисунок 3.7).

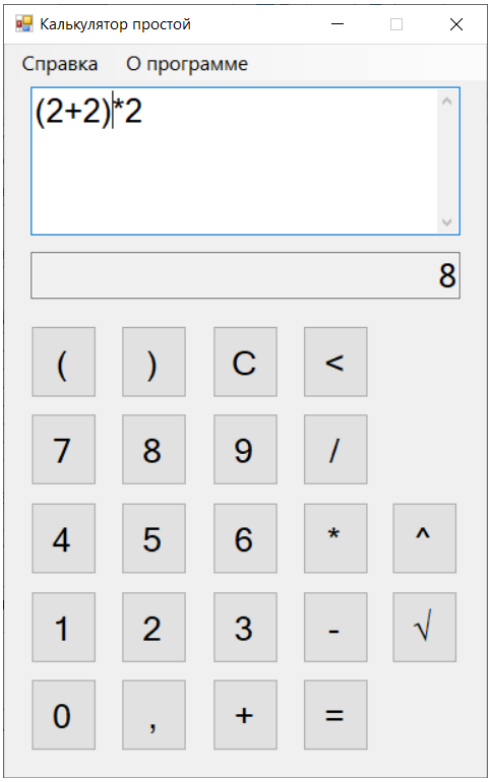

Рисунок 3.6 – Редактирование выражения

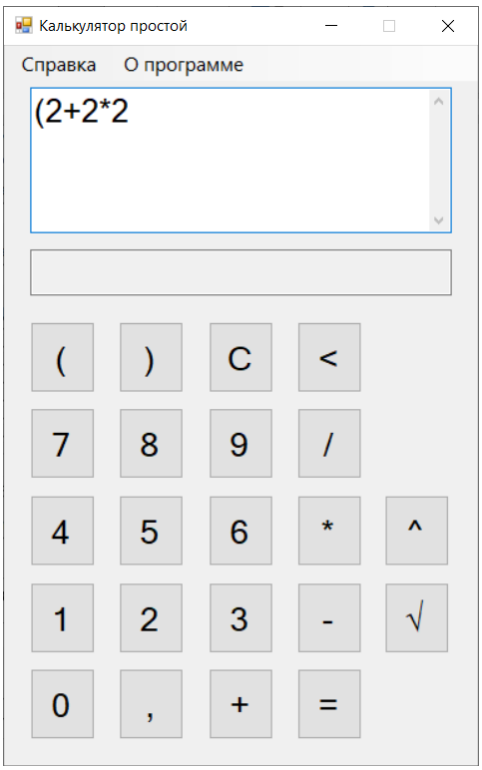

Рисунок 3.7 – Некорректное выражение

Использование радикала в двух ситуациях (рисунки 3.8 и 3.9).

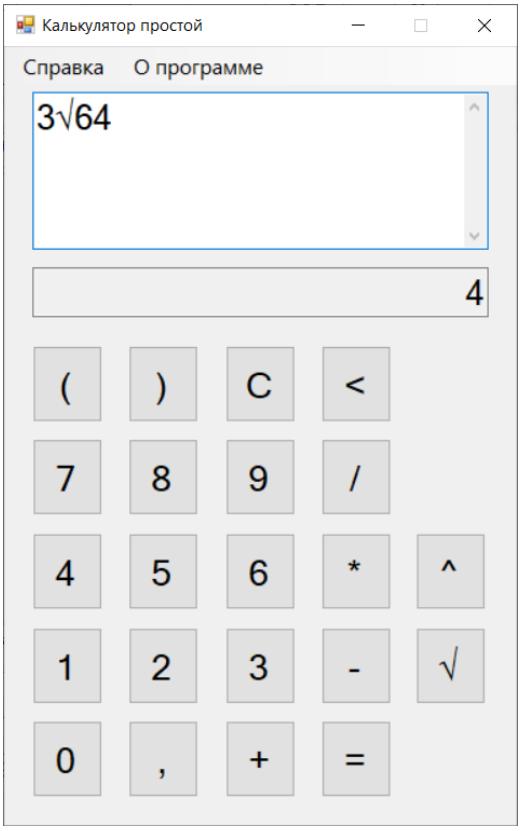

Рисунок 3.8 – Как корень третьей степени

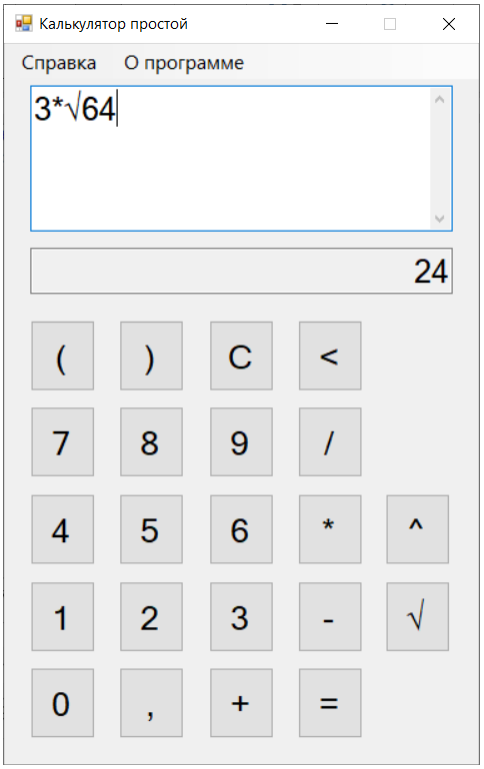

Рисунок 3.9 – Умножение выражения на квадратный корень

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были выбраны оптимальные визуальные компоненты и успешно внедрены в алгоритм решения. Разработанный код был проверен на контрольных тестах и в код были внесены необходимые исправления. Для приложения была разработана документация, описывающая ее установку и использование. Таким образом, цель работы была достигнута, задачи – решены.

Результаты работы можно использовать в процессе последующего обучения в форме навыков практического применения визуального программирования для разработки сложных программных систем, понимания порядка этапов разработки программного обеспечения и достигаемых на каждом этапе результатов.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Свердлов, С. З. Языки программирования и методы трансляции : учебное пособие / С. З. Свердлов. — 2-е изд., испр. — Санкт-Петербург : Лань, 2019. — 564 с. — ISBN 978-5-8114-3457-2. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/116391 (дата обращения: 03.05.2021). — Режим доступа: для авториз. пользователей.

2. Тюкачев, Н. А. C#. Основы программирования : учебное пособие для спо / Н. А. Тюкачев, В. Г. Хлебостроев. — Санкт-Петербург : Лань, 2021. — 272 с. — ISBN 978-5-8114-6816-4. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/154116 (дата обращения: 21.05.2021). — Режим доступа: для авториз. пользователей.

3. Мурадханов, С. Э. Разработка на языке C# приложений с графическим интерфейсом (использование Windows Forms) : учебник / С. Э. Мурадханов. — Москва : МИСИС, 2019. — 396 с. — ISBN 978-5-907061-36-1. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/129040 (дата обращения: 21.05.2021). — Режим доступа: для авториз. пользователей.

Исходный текст программы

Файл Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace SimpleCalculator
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Файл 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 SimpleCalculator
{
    public partial class Form1 : Form
    {
        private Calculation.TypesErros error =
Calculation.TypesErros.NoError;
        private char codeKey = (char)0;
        private int selectedInput = 0;
```

```
        public Form1()
        {
            InitializeComponent();
        }

        private void AddInputTextBox(string s)
        {
            codeKey = '0';
            selectedInput = inputFieldTextBox.SelectionStart;
            inputFieldTextBox.Text =
inputFieldTextBox.Text.Insert(selectedInput, s);
        }

        // здесь должно вычисляться выражение
        private void Calulate()
        {
            if (inputFieldTextBox.TextLength == 0)
            {
                return;
            }
            outputFieldTextBox.Text =
                Calculation.Calculate(inputFieldTextBox.Text,
out error);
        }

        private void Back(bool isButton)
        {
            codeKey = (char)0;
            if (inputFieldTextBox.TextLength == 0)
            {
                outputFieldTextBox.Text = "";
                return;
            }
            if (isButton)
            {
                codeKey = (char)0;
                selectedInput =
inputFieldTextBox.SelectionStart;
                if (selectedInput > 0)
                {
                    inputFieldTextBox.Text =

inputFieldTextBox.Text.Remove(selectedInput - 1, 1);
                }
                if (!inputFieldTextBox.Focused)
                {
                    inputFieldTextBox.SelectionStart =
(selectedInput > 0 ? selectedInput - 1 : 0);
                    inputFieldTextBox.Focus();
                }

            }
```

```
            if (inputFieldTextBox.TextLength != 0)
            {
                Calulate();
            }
            else
            {
                outputFieldTextBox.Text = "";
            }
        }

        private void Drop()
        {
            codeKey = (char)0;
            inputFieldTextBox.Text = outputFieldTextBox.Text =
"";
            if (!inputFieldTextBox.Focused)
            {
                inputFieldTextBox.SelectionStart = 0;
                inputFieldTextBox.Focus();
            }
        }

        private void Finish()
        {
            if (error != Calculation.TypesErros.NoError)
            {
                string text = "";
                switch (error)
                {
                    case Calculation.TypesErros.Lexical:
                        text = "Лексическая ошибка";
                        break;
                    case Calculation.TypesErros.Syntax:
                        text = "Синтаксическая ошибка";
                        break;
                    case Calculation.TypesErros.DivideByZero:
                        text = "Деление на ноль";
                        break;
                    case Calculation.TypesErros.RootNegative:
                        text = "Некорректные параметры корня";
                        break;
                    default:
                        text = "Неизвестная ошибка";
                        break;
                }
                MessageBox.Show(text, "Ошибка",
MessageBoxButtons.OK);
                return;
            }
            inputFieldTextBox.Text = outputFieldTextBox.Text;
            inputFieldTextBox.SelectionStart = 0;
```

```
            inputFieldTextBox.SelectionLength =
inputFieldTextBox.TextLength;
            inputFieldTextBox.Focus();
        }

        private void inputFieldTextBox_KeyPress(object sender,
KeyPressEventArgs e)
        {
            if (Char.IsDigit(e.KeyChar) || e.KeyChar ==
(char)Keys.Back ||
                    e.KeyChar == (char)Keys.Enter || e.KeyChar
== (char)Keys.Delete ||
                    e.KeyChar == '+' || e.KeyChar == '-' ||
e.KeyChar == '*' ||
                    e.KeyChar == '/' || e.KeyChar == ',' ||
e.KeyChar == '.' ||
                    e.KeyChar == '(' || e.KeyChar == ')' ||
e.KeyChar == '^' ||
                    e.KeyChar == '\\')
            {
                if (e.KeyChar == '.')
                {
                    e.KeyChar = ',';
                }
                if (e.KeyChar == '\\')
                {
                    e.KeyChar = '√';
                }
                codeKey = e.KeyChar;
            }
            else
            {
                e.Handled = true;
            }
        }

        private void button0_Click(object sender, EventArgs e)
        {
            if (sender is Button)
            {
                AddInputTextBox(((Button)sender).Text);
            }
        }

        private void finishButton_Click(object sender, EventArgs
e)
        {
            Finish();
        }
```

```
        private void dropButton_Click(object sender, EventArgs
e)
        {
            Drop();
        }

        private void backspaceButton_Click(object sender,
EventArgs e)
        {
            Back(true);
        }

        private void справкаToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            string text =
                "3√64=4 (корень кубический)" +
Environment.NewLine +
                "3*√64=3*8=24 (корень квадратный)" +
Environment.NewLine +
                "Чтобы ввести '√' с клавиатуры, нажмите '\\'";
            MessageBox.Show(text, "Справка",
MessageBoxButtons.OK);
        }

        private void оПрограммеToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            string text =
                "Автор программы" + Environment.NewLine +
                "Чумаков Денис" + Environment.NewLine +
                "Группа ЕТ-312" + Environment.NewLine +
                "2021";
            MessageBox.Show(text, "О программе",
MessageBoxButtons.OK);
        }

        private void inputFieldTextBox_KeyDown(object sender,
KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Delete)
            {
                Drop();
            }
        }
```

```
        private void inputFieldTextBox_TextChanged(object
sender, EventArgs e)
        {
            if (codeKey == (char)0)
            {
                return;
            }

            if (codeKey == (char)Keys.Back)
            {
                Back(false);
            }
            else if (codeKey == (char)Keys.Enter)
            {
                codeKey = (char)0;
                int count = Environment.NewLine.Length;
                inputFieldTextBox.Text =

inputFieldTextBox.Text.Remove(inputFieldTextBox.TextLength -
count, count);
                Finish();
            }
            else
            {
                codeKey = (char)0;
                Calulate();
                if (!inputFieldTextBox.Focused)
                {
                    inputFieldTextBox.SelectionStart =
selectedInput + 1;
                    inputFieldTextBox.Focus();
                }
            }
        }
    }
}
```

Файл Calculation.cs

```
using System;
using System.Collections.Generic;

namespace SimpleCalculator
{
    public static class Calculation
    {
        // 1-низкий приоритет, 3-высокий
        // унарные операции еще выше
        private static Dictionary<string, int> priotities =
            new Dictionary<string, int>() {
            {"+", 1},
            {"-", 1}, // бинарный минус
```

```
            {"*", 2},
            {"/", 2},
            {"^", 3}, // степень 3^4=81
            {"n√", 3}, //корень 4√81=3
            {"(", 0},
            {")", 0},
            {"+-", 0}, // унарный минус
            {"√", 0} // квадратный корень √25=5
        };
        class Term
        {
            public enum TERM { NUMBER, OPERATION };

            public TERM type;
            public string operation;
            public double number;
            public Term()
            {
                type = TERM.NUMBER;
                operation = "";
                number = 1.0;
            }
            public Term(double _number)
            {
                type = TERM.NUMBER;
                number = _number;
            }
            public Term(string op)
            {
                type = TERM.OPERATION;
                operation = op;
            }
        }

        public enum TypesErros { NoError, Lexical, Syntax,
DivideByZero, RootNegative };

        public static string Calculate(string expression, out
TypesErros error)
        {
            List<Term> terms = ReadTerms(expression, out error);
            if (error != TypesErros.NoError)
            {
                return "";
            }

            List<Term> poliz = ToPoliz(terms, out error);
            if (error != TypesErros.NoError)
            {
                return "";
            }
```

```
            double answer = Calc(poliz, out error);
            if (error != TypesErros.NoError)
            {
                return "";
            }
            return answer.ToString();
        }
        private static double Calc(List<Term> poliz, out
TypesErros error)
        {
            // вычисление значения по обратной польской записи
            error = TypesErros.NoError;
            Stack<double> stack = new Stack<double>();
            foreach (Term term in poliz)
            {
                if (term.type == Term.TERM.NUMBER)
                {
                    stack.Push(term.number);
                }
                else if (IsUnaryOperation(term))
                {
                    if (stack.Count == 0)
                    {
                        error = TypesErros.Syntax;
                        return Double.NaN;
                    }

                    double top = stack.Pop();
                    double resultOperation = Operate(top,
term.operation, out error);
                    if (error != TypesErros.NoError)
                    {
                        return Double.NaN;
                    }
                    stack.Push(resultOperation);
                }
                else
                {
                    if (stack.Count < 2)
                    {
                        error = TypesErros.Syntax;
                        return Double.NaN;
                    }
                    double secondOperand = stack.Pop();
                    double firstOperand = stack.Pop();

                    double resultOperation =
Operate(firstOperand, term.operation,
                            secondOperand, out error);
```

```
                    if (error != TypesErros.NoError)
                    {
                        return Double.NaN;
                    }
                    stack.Push(resultOperation);
                }
            }
            if (stack.Count != 1)
            {
                error = TypesErros.Syntax;
                return Double.NaN;
            }
            return stack.Peek();
        }

        private static double Operate(double operand, string
operation, out TypesErros error)
        {
            // унарная операция
            error = TypesErros.NoError;
            switch (operation)
            {
                case "+-":
                    return -operand;
                case "√":
                    if (operand < Double.Epsilon)
                    {
                        error = TypesErros.RootNegative;
                        return Double.NaN;
                    }
                    return Math.Sqrt(operand);
                default:
                    error = TypesErros.Syntax;
                    return Double.NaN;
            }
        }

        private static double Operate(double firstOperand,
string operation, double secondOperand, out TypesErros error)
        {
            // бинарная операция
            error = TypesErros.NoError;
            switch (operation)
            {
                case "+":
                    return firstOperand + secondOperand;
                case "-":
                    return firstOperand - secondOperand;
                case "*":
                    return firstOperand * secondOperand;
```

```
                case "/":
                    if (Math.Abs(secondOperand) <
Double.Epsilon)
                    {
                        error = TypesErros.DivideByZero;
                        return Double.NaN;
                    }
                    return firstOperand / secondOperand;
                case "^":
                    return Math.Pow(firstOperand,
secondOperand);
                case "n√":
                    if (firstOperand < Double.Epsilon)
                    {
                        error = TypesErros.RootNegative;
                        return Double.NaN;
                    }
                    if (secondOperand < Double.Epsilon)
                    {
                        int tr =
(int)Math.Truncate(firstOperand);
                        if (tr % 2 != 0 && Math.Abs(tr -
firstOperand) < Double.Epsilon)
                        {
                            return -Math.Pow(-secondOperand,
1/firstOperand);
                        }
                    }
                    return Math.Pow(secondOperand,
1/firstOperand);
                default:
                    error = TypesErros.Syntax;
                    return Double.NaN;
            }
        }

        private static List<Term> ToPoliz(List<Term> terms, out
TypesErros error)
        {
            // преобразование из инфиксной нотации в
            // обратную польскую
            error = TypesErros.NoError;
            List<Term> poliz = new List<Term>();
            Stack<Term> stack = new Stack<Term>();
            foreach (Term term in terms)
            {
                if (term.type == Term.TERM.NUMBER)
                {
                    poliz.Add(term);
                }
```

```
                else if (IsUnaryOperation(term) ||
term.operation == "(")
                {
                    stack.Push(term);
                }
                else if (term.operation == ")")
                {
                    bool isOpenParanthes = false;
                    while (stack.Count != 0)
                    {
                        if (stack.Peek().operation == "(")
                        {
                            isOpenParanthes = true;
                            stack.Pop();
                            break;
                        }
                        poliz.Add(stack.Pop());
                    }
                    if (!isOpenParanthes)
                    {
                        error = TypesErros.Syntax;
                        return null;
                    }
                }
                else
                {
                    while (stack.Count != 0 &&
(IsUnaryOperation(stack.Peek()) ||
                            priotities[stack.Peek().operation] >
priotities[term.operation] ||
                            priotities[stack.Peek().operation]
== priotities[term.operation] &&
                                IsLeftAssoc(stack.Peek())))
                    {
                        poliz.Add(stack.Pop());
                    }
                    stack.Push(term);
                }
            }
            while (stack.Count != 0) {
                if (stack.Peek().operation == "(" ||
                    stack.Peek().operation == ")")
                {
                    error = TypesErros.Syntax;
                    return null;
                }
                poliz.Add(stack.Pop());
            }
            return poliz;
        }
```

```
        private static bool IsUnaryOperation(Term t)
        {
            return t.operation == "+-" || t.operation == "√";
        }

        private static bool IsLeftAssoc(Term t)
        {
            return t.operation == "+" || t.operation == "-" ||
                t.operation == "*" || t.operation == "/";
        }

        private static List<Term> ReadTerms(string expression,
out TypesErros error)
        {
            // разбор исходного выражения на лексемы-токены
            error = TypesErros.NoError;
            List<Term> terms = new List<Term>();
            bool isNumber = false;
            string word = "";
            for (int i = 0; i < expression.Length; i++)
            {
                if (Char.IsDigit(expression[i]) || expression[i]
== ',')
                {
                    isNumber = true;
                    word += expression[i];
                }
                else
                {
                    if (isNumber)
                    {
                        double number;
                        if (Double.TryParse(word, out number))
                        {
                            terms.Add(new Term(number));
                            word = "";
                            isNumber = false;
                        }
                        else
                        {
                            error = TypesErros.Lexical;
                            return null;
                        }
                    }
                    int t;
                    word += expression[i];
```

```
                    if (priotities.TryGetValue(word, out t))
                    {
                        if (word == "-" && (terms.Count == 0 ||
                                terms[terms.Count - 1].type ==
Term.TERM.OPERATION &&
                                terms[terms.Count - 1].operation
!= ")"))
                        {
                            word = "+-";
                        }
                        if (word == "√" && terms.Count != 0 &&
                                (terms[terms.Count - 1].type ==
Term.TERM.NUMBER ||
                                terms[terms.Count - 1].operation
== ")"))
                        {
                            word = "n√";
                        }
                        terms.Add(new Term(word));
                        word = "";
                    }
                    else
                    {
                        error = TypesErros.Lexical;
                        return null;
                    }
                }
            }
            if (isNumber)
            {
                double number;
                if (Double.TryParse(word, out number))
                {
                    terms.Add(new Term(number));
                }
                else
                {
                    error = TypesErros.Lexical;
                    return null;
                }
            }
            return terms;
        }
    }
}
```