МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра автоматизированных систем управления



ОТЧЁТ

по КУРСОВОЙ РАБОТЕ

**«**Объявление комплексного числа с инициализацией на языке C#**»**

по дисциплине: **«***Теория формальных языков и компиляторов***»**

Выполнил:Проверил:

Студент гр. «*АВТ-214*», «*АВТФ*» *д.т.н., профессор*

*Ядров Максим Владимирович Шорников Юрий Владимирович*

«\_\_\_» \_\_\_\_\_\_ 20\_\_г.«\_\_\_» \_\_\_\_\_\_ 20\_\_ г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (подпись)

Новосибирск 2025

**РЕФЕРАТ**

Отчет 36 с., 14 рис., 3 источн., 3 прилож.

ЯЗЫКОВОЙ ПРОЦЕССОР, КОМПИЛЯТОР, ЛЕКСИЧЕСКИЙ АНАЛИЗ, СИНТАКСИЧЕСКИЙ АНАЛИЗ, АВТОМАТНАЯ ГРАММАТИКА, ГРАФ АВТОМАТНОЙ ГРАММАТИКИ, ДИАГНОСТИКА И НЕЙТРАЛИЗАЦИЯ ОШИБОК, МЕТОД АЙРОНСА

Цель работы – выполнить программную реализацию объявления и инициализации комплексного числа на языке C#.

В результате проектирования был написан компилятор на языке C#, который работает с файлами и проверяет с помощью анализатора и парсера правильность введённого текста, в соответствии с заданной грамматикой.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc193676120)

[I. Постановка задачи 5](#_Toc193676121)

[II. Разработка грамматики 6](#_Toc193676122)

[III. Классификация грамматики 7](#_Toc193676123)

[IV. Метод анализа 8](#_Toc193676124)

[V. Диагностика и нейтрализация синтаксических ошибок 9](#_Toc193676125)

[VI. Тестовые примеры 10](#_Toc193676126)

[VII. Листинг программы 16](#_Toc193676127)

[ЗАКЛЮЧЕНИЕ 17](#_Toc193676128)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 18](#_Toc193676129)

[Приложение А 19](#_Toc193676130)

[Справка (руководство пользователя) 19](#_Toc193676131)

[Меню текстового редактора 19](#_Toc193676132)

[Пункт "Файл" меню текстового редактора 19](#_Toc193676133)

[Пункт "Правка" меню текстового редактора 19](#_Toc193676134)

[Пункт "Текст" меню текстового редактора 20](#_Toc193676135)

[Пункт "Пуск" меню текстового редактора 21](#_Toc193676136)

[Пункт "Справка" меню текстового редактора 22](#_Toc193676137)

[Панель инструментов текстового редактора 22](#_Toc193676138)

[Приложение Б 23](#_Toc193676139)

[Информация о программе 23](#_Toc193676140)

[Приложение В 24](#_Toc193676141)

[Исходный код программы 24](#_Toc193676142)

# ВВЕДЕНИЕ

Цель курсовой работы – выполнить программную реализацию объявления комплексного числа с инициализацией на языке C#.

Курсовая работа содержит следующие разделы:

* Постановка задачи;
* Грамматика;
* Классификация грамматики;
* Метод анализа;
* Диагностика и нейтрализация ошибок;
* Тестовый пример;
* Список литературы;
* Исходный код программы.

# I. Постановка задачи

В C# для объявления и инициализации комплексного числа используется структура System.Numerics.Complex. Эта структура позволяет представить комплексное число как пару чисел с плавающей точкой, представляющих действительную и мнимую части. После объявления переменной типа Complex, ей можно присвоить значение, представляющее комплексное число, которое в дальнейшем не будет изменяться (хотя сама переменная может быть переназначена).

Формат записи: “Complex $name = new Complex ($value, $value);”

Параметр $name передает имя переменной, а параметр $value - ее значения. Значение может быть целым (int), дробным (float, double, exp).

Примеры:

1. Complex Perem = new Complex (2, 0);
2. Complex Perem = new Complex (3.4, -5.7f);
3. Complex Perem = new Complex (1.23e-4f, 1.23e-4f);

# II. Разработка грамматики

Определим грамматику объявления и инициализации комплексного числа на языке C# G[<DEF>] в нотации Хомского с продукциями P:

1. <DEF> 🡪 ‘Complex’**•**IDENT
2. IDENT 🡪 letter**•**IDREM
3. IDREM 🡪 letter **•** IDREM | digit**•** IDREM| \_IDREM| ‘=’NEW
4. NEW 🡪 ‘new’**•TYPE**
5. **TYPE 🡪 ‘Complex’•CONSTRUCTOR**
6. **CONSTRUCTOR 🡪 ‘(’ •SIGN**
7. **SIGN 🡪 ‘-’ •NUMBERS | NUMBERS**
8. **NUMBERS 🡪 digit • NUM**
9. **NUM 🡪 digit • REAL | ‘.’REAL | ‘,’ IMAGINARY**
10. **REAL 🡪 digit • REALREM**
11. **REALREM 🡪 digit•REALREM | ‘,’IMAGINARYSIGN**
12. **IMAGINARYSIGN 🡪 IMAGINARY | ‘-’ • IMAGINARY**
13. **IMAGINARY 🡪 digit • INTIMAGINARY**
14. **INTIMAGINARY 🡪 digit • INTIMAGINARY| ‘.’INTIMAGINARYREM | ‘)’ •END**
15. **INTIMAGINARYREM 🡪 digit • INTIMAGINARYREM | ‘)’ •END**
16. **END 🡪 ‘;’**
17. <letter> → "a" | "b" | "c" | ... | "z" | "A" | "B" | "C" | ... | "Z" |  
    <digit> → "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Следуя введенному формальному определению грамматики, представим G[<DEF>] ее составляющими:

* Z = G[<DEF >] ;
* VT = {a, b, c, ..., z, A, B, C, ..., Z, +, -, ;, ,, .,0, 1, 2, ..., 9};
* VN = {<DEF>, <IDENT>, <IDREM>, <NEW>, <TYPE>, <CONSTRUCTOR>, <SIGN>, <NUMBERS>,<NUM>, <REAL>, <REALREM>, < **IMAGINARYSIGN** >, <IMAGINARY>, <INTIMAGINARY>, <IMAGINARYREM>, <END>}

# III. Классификация грамматики

Н. Хомский выделил четыре класса грамматик: грамматики нулевого типа, контекстно-зависимые, контекстно-свободные и автоматные грамматики. Вид грамматики определяется исходя из формы записи ее правил.

Грамматики нулевого типа имеют правила следующего вида:

α→β,

где α ∈ V\*, V = Vт ⋃ VN, β ∈ V\*.

Контекстно-зависимые грамматики (КЗграмматики) имеют правила следующего вида:

γ1Aγ2 → γ1βγ2,

где A ∈ VN, β∈V+, γ1∈V\* и γ2∈V\*.

Контекстно-свободные грамматики имеют правила следующего вида:

A → a,

где A∈VN, α∈V\*.

Автоматные или регулярные грамматики имеют самые строгие ограничения на форму записи правил:

A → aB | a | ε,

где a∈VТ, A∈VN и B∈VB.

Согласно приведённым выше правилам, грамматика G[<DEF>] является автоматной грамматикой, т.к. здесь в левой части правил находится только по одному нетерминальному символу, а в правой – либо один терминальный и один нетерминальный символ, либо один терминальный символ.

# IV. Метод анализа

Для анализа грамматики было выбрано её представление в виде графа конечного автомата (рис. 1), где состояние 1 – является начальным, а состояние 15 – конечным.

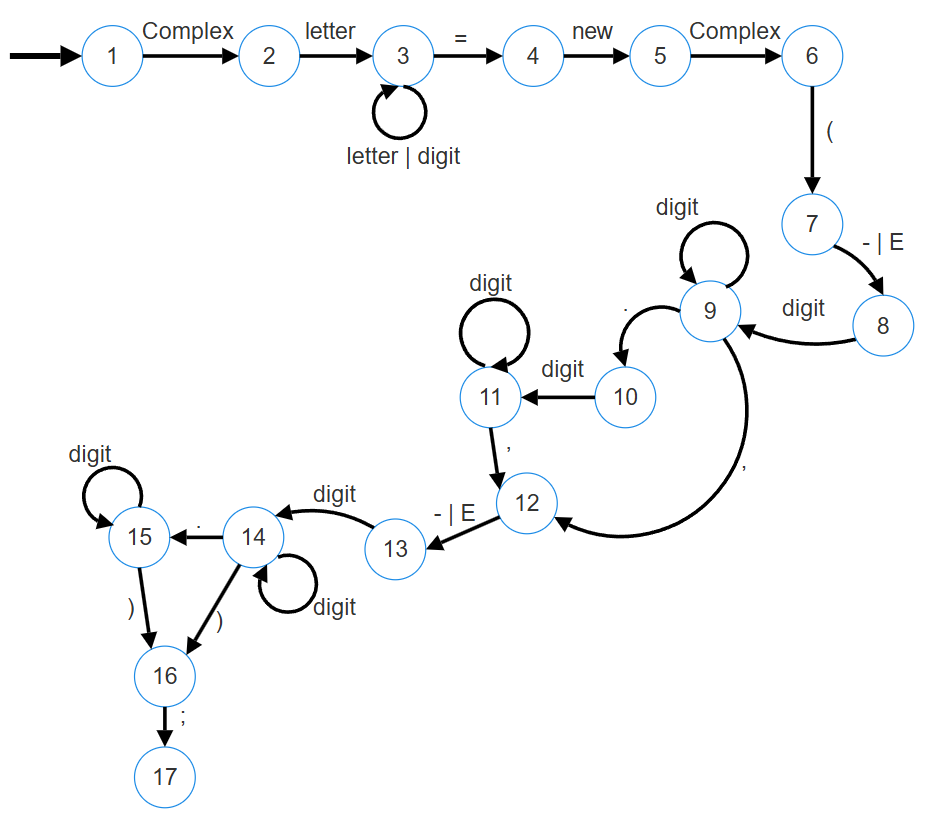


Рисунок 1 – Граф конечного автомата грамматики G[‹DEF›]

# V. Диагностика и нейтрализация синтаксических ошибок

Для нашей программы необходимо реализовать нейтрализацию ошибок.

Проверка производится поэтапно:

1. Сначала программа посимвольно проверяет строку на наличие запрещённых символов. Программа помечает “проблемные” места, где находятся подстроки запрещённых символов, затем удаляет их и соединяет оставшиеся подстроки. Все удалённые подстроки заносятся в список ошибок.
2. Программа возвращается в начало строки и движется уже не по символам, а по токенам. Токеном может быть число, символ и слово. Далее программа проходит по токенам и сравнивает полученный с ожидаемым. Если они совпали, то программа идёт дальше. Если нет, то: программа идёт до конца строки в поисках нужного токена. Если такой найден, то все токены между ним и предыдущим обязательным отбрасываются с пометкой об их ошибочности. В противном случае программа обозначит искомый токен пропущенные и перейдёт к поиску следующего.

# VI. Тестовые примеры

На рисунках 2-7 представлены тестовые примеры запуска разработанного синтаксического анализатора объявления и инициализации комплексного числа на языке C#.

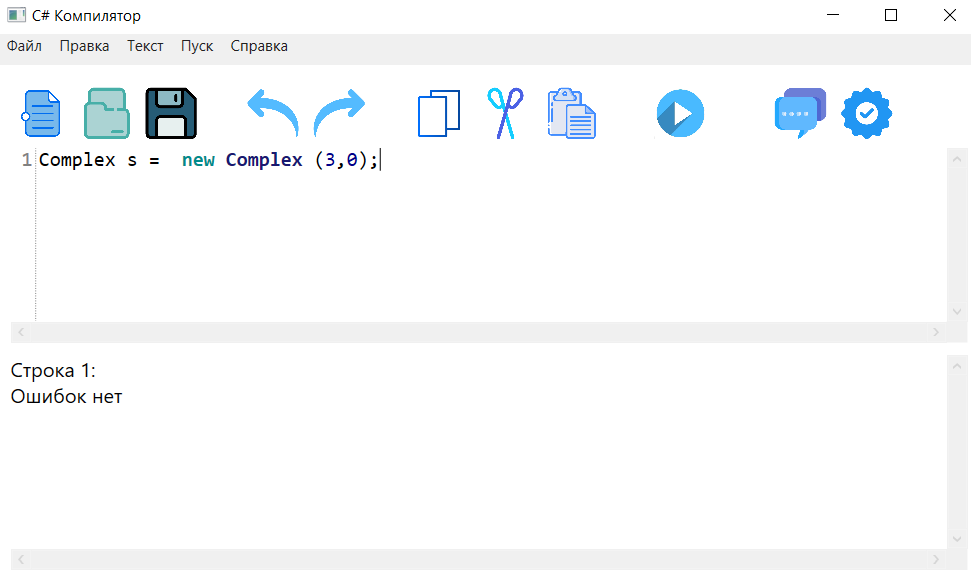


Рисунок 2 – Тестовый пример 1

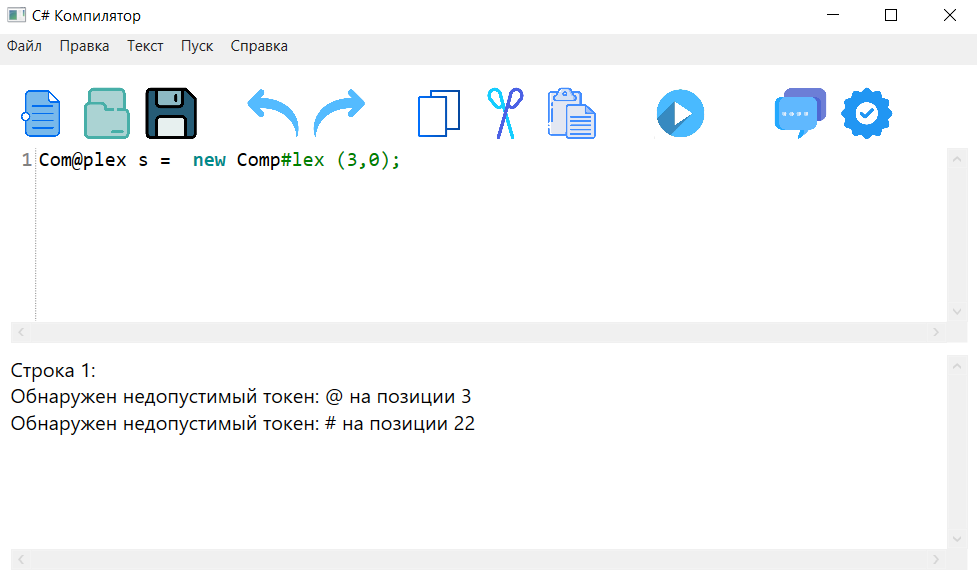


Рисунок 3 – Тестовый пример 2

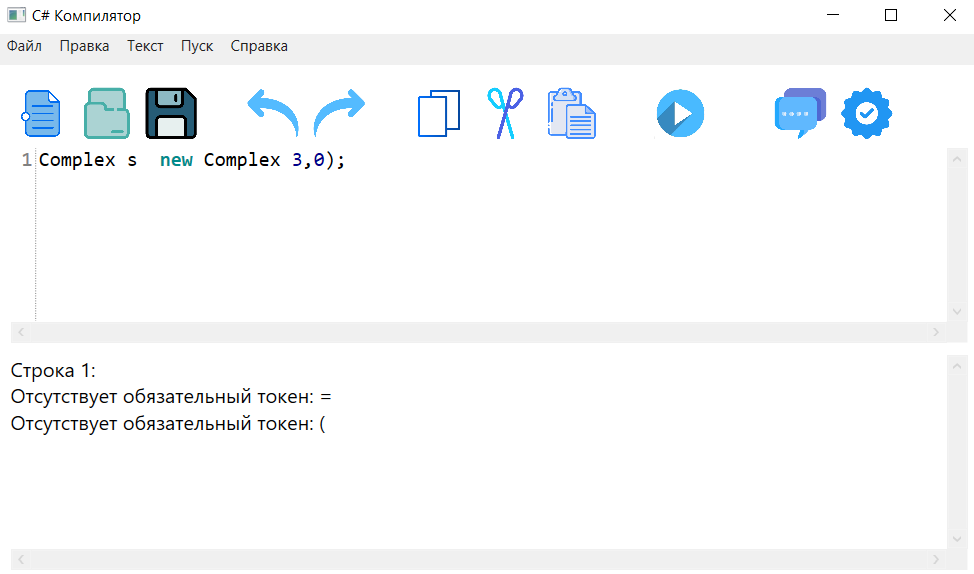


Рисунок 4 – Тестовый пример 3

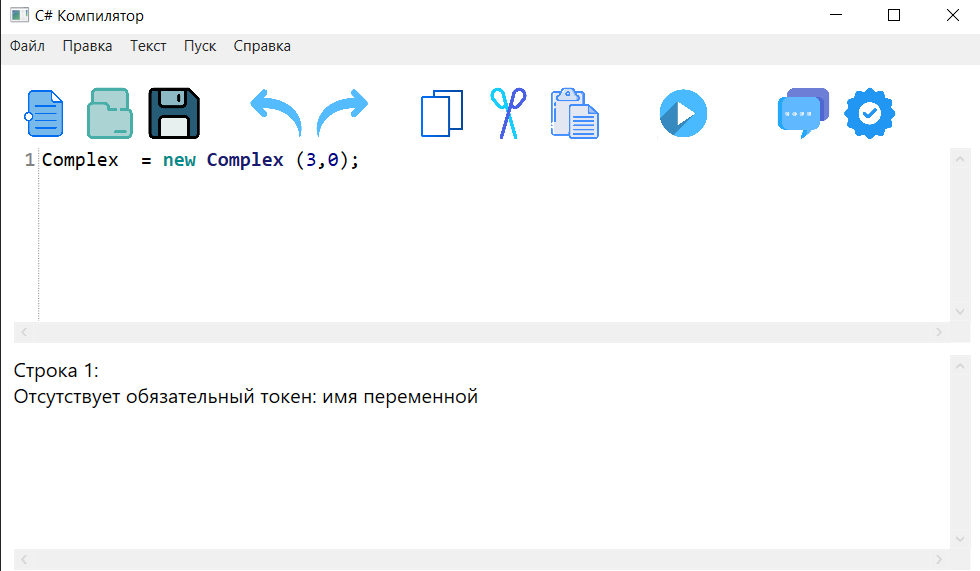


Рисунок 5 – Тестовый пример 4

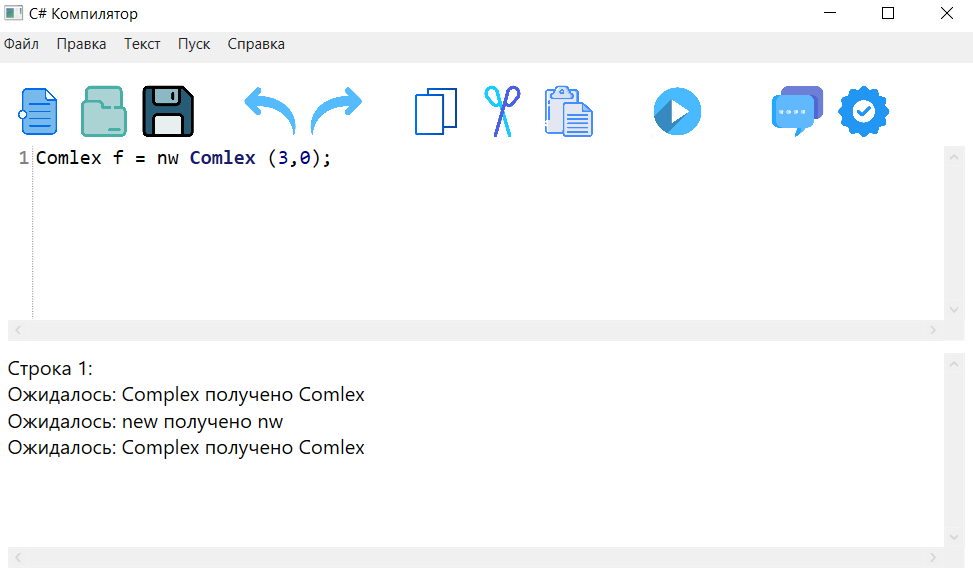


Рисунок 6 – Тестовый пример 5

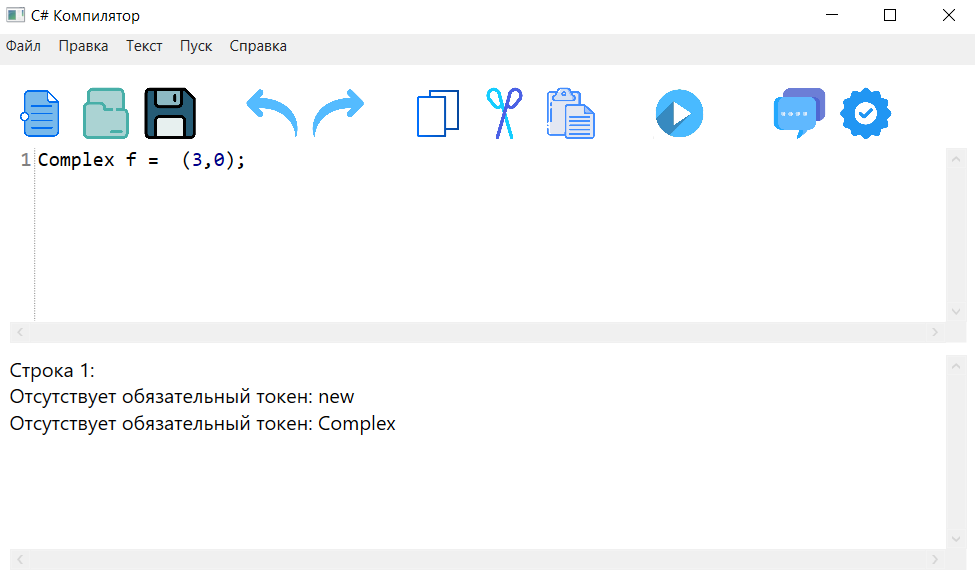


Рисунок 7 – Тестовый пример 6

# VII. Листинг программы

Листинг исходного кода разработанного синтаксического анализатора объявления и инициализации комплексного числа на языке C# представлен в приложении В.

# ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы были разработаны грамматика G[DEF] и конечный автомат, описывающие процесс объявления и инициализации комплексного числа в языке C#. Создана консольная программа, способная обрабатывать текстовые файлы, в которую интегрированы лексический анализатор, парсер и модуль нейтрализации синтаксических ошибок, основанный на методе Айронса. Стратегия нейтрализации ошибок заключалась в последовательном удалении символов из входной строки до тех пор, пока следующий символ не станет допустимым в текущем контексте синтаксического анализа объявления комплексного числа.

В процессе работы были выявлены основные проблемы и трудности при создании собственных лексических и синтаксических анализаторов для C#, а также разработаны эффективные методы и инструменты для их решения.

Таким образом, результаты работы имеют значительную практическую ценность, поскольку полученные знания и навыки могут быть применены при работе с компиляторами C# и разработке специализированных инструментов для анализа кода, работающего с комплексными числами.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шорников Ю.В. [Теория языков программирования: проектирование и реализация : учебное пособие / Ю. В. Шорников. – Новосибирск: Изд-во НГТУ, 2022. – 290 с.](https://elibrary.nstu.ru/source?id=178996)
2. Биллиг, В. А. Основы программирования на С# / В.А. Биллиг. - М.: Интернет-университет информационных технологий, Бином. Лаборатория знаний, **2021**. - 488 c.
3. Вагнер, Билл С# Эффективное программирование / Билл Вагнер. - М.: ЛОРИ, **2021**. - 320 c.

# Приложение А

# Справка (руководство пользователя)

## Меню текстового редактора

### Пункт "Файл" меню текстового редактора

В пункте "Файл" меню текстового редактора реализован следующий функционал (см. рисунок А.1):

* Создать новый докумет
* Открыть документ
* Сохранить текущий документ
* Сохранить текущий документ в определённое место
* Выйти из программы

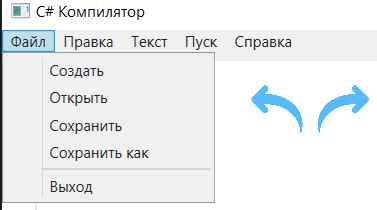


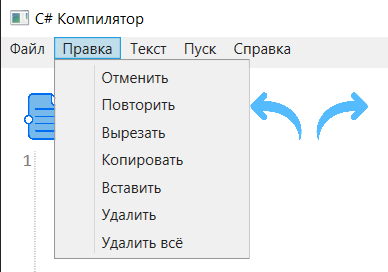
Рисунок А.1 – Пункт "Файл" меню

### Пункт "Правка" меню текстового редактора

В пункте "Правка" меню текстового редактора находится функционал для правки введённого текста (см. рисунок А.2):

* Отмена изменений
* Повтор последнего изменения
* Вырезать выделенный текстовый фрагмент
* Копировать выделенный текстовый фрагмент
* Вставить текстовый фрагмент
* Удалить выделенный текстовый фрагмент
* Выделить все содержимое документа

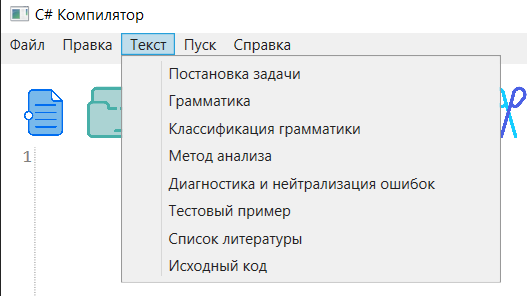
Рисунок А.2 – Пункт "Правка" меню



### Пункт "Текст" меню текстового редактора

Пункт меню "Текст" содержит команды для вывода информации о курсовой работе и программы (см. рисунок А.3):

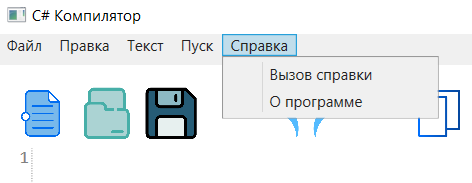
* Постановка задачи
* Грамматика
* Классификация грамматики
* Метод анализа
* Диагностика и нейтрализация ошибок
* Тестовый пример
* Список литературы
* Исходный код программы

  
Рисунок А.3 – Пункт "Текст" меню

### Пункт "Пуск" меню текстового редактора

При нажатии на пункт "Пуск" происходит запуск парсера(см. рисунок А.4).

Рисунок А.4 – Пункт "Пуск" меню



### Пункт "Справка" меню текстового редактора

Пункт меню "Справка" содержит подпункт "Вызов справки" -справочное описание программы и подпункт "О программе" - версия и информация о разработчике (см. рисунок А.5)

Рисунок А.5 – Пункт "Справка" меню

## Панель инструментов текстового редактора

Панель инструментов содержит кнопки вызова часто используемых пунктов меню:

* Создание документа
* Открытие документа
* Сохранение текущих изменений в документе
* Отмена изменений
* Повтор последнего изменения
* Копировать выделенный текстовый фрагмент
* Вырезать выделенный текстовый фрагмент
* Вставить текстовый фрагмент

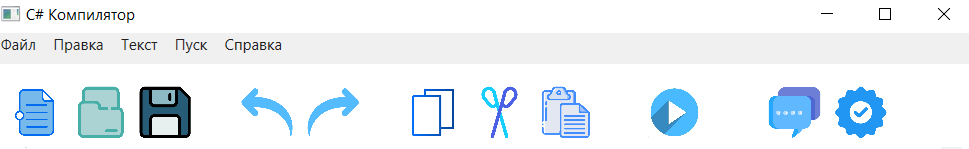


Рисунок А.6 – Панель инструментов

# Приложение Б

# Информация о программе

Программа может работать с текстовыми файлами типа txt. Для этого у пользователя есть функции для работы с файлами:

* Создать новый файл;
* Открыть существующий файл;
* Сохранить файл;
* Сохранить как новый файл;
* Выход - завершает программу.

Для работы с текстом для пользователя реализованы следующие функции:

* Отменить предыдущее изменение;
* Повторить изменение;
* Вырезать выделенный текст в буфер обмена;
* Копировать выделенный текст в буфер обмена;
* Вставить текст из буфера обмена;
* Удалить выделенный текст;
* Выделить весь текст из области редактирования файла.

Приложение имеет справочную систему, запускаемую командой «Вызов справки».

В область отображения результатов выводятся сообщения и результаты работы языкового процессора.

Кнопка меню пуск запускает парсер, который анализирует введённый текс.

При этом выводятся все допущенные ошибки.

# Приложение В

# Исходный код программы

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.IO;

using System.Text;

using System.Text.RegularExpressions;

using System.Windows.Media;

using Compiler;

using System.Windows.Threading;

using System.Diagnostics;

using System.ComponentModel;

using System;

using System.Collections.Generic;

using System.Windows.Input;

using Microsoft.Win32;

using Path = System.IO.Path;

using Color = System.Windows.Media.Color;

using Clipboard = System.Windows.Forms.Clipboard;

using TextDataFormat = System.Windows.Forms.TextDataFormat;

using MessageBox = System.Windows.Forms.MessageBox;

using Application = System.Windows.Application;

using OpenFileDialog = Microsoft.Win32.OpenFileDialog;

using System.Windows.Forms;

namespace Compilator

{

public partial class MainWindow : Window

{

private string fileName;

private string filePath;

private string filesFolderPath;

private DispatcherTimer timer;

public MainWindow()

{

InitializeComponent();

string baseDirectory = AppDomain.CurrentDomain.BaseDirectory;

string projectDirectory = Directory.GetParent(Directory.GetParent(baseDirectory).FullName).FullName;

filesFolderPath = Path.Combine(Directory.GetParent(projectDirectory).FullName, "Files");

timer = new DispatcherTimer();

timer.Interval = TimeSpan.FromMilliseconds(200);

timer.Tick += Timer\_Tick;

}

private void Timer\_Tick(object sender, EventArgs e)

{

timer.Stop();

ColorizeRichTextBox(RCB1);

}

private void RichTextBox\_TextChanged(object sender, TextChangedEventArgs e)

{

}

private void MakeNewFile(object sender, RoutedEventArgs e)

{

CreateWindow createFileWindow = new CreateWindow();

createFileWindow.Closed += CreateFileDialog\_Closed;

createFileWindow.ShowDialog();

}

private void CreateFileDialog\_Closed(object sender, EventArgs e)

{

fileName = ((CreateWindow)sender).FileName;

filePath = Path.Combine(filesFolderPath, fileName);

if (!Directory.Exists(filesFolderPath))

{

try

{

Directory.CreateDirectory(filesFolderPath);

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при создании папки Files: " + ex.Message);

return;

}

}

try

{

File.WriteAllText(filePath, "Hello");

textEditor.Clear();

textEditor.Text += "Hello";

MessageBox.Show("Файл успешно создан: " + fileName);

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при создании файла: " + ex.Message);

}

}

private void SaveFile\_Click(object sender, RoutedEventArgs e)

{

try

{

File.WriteAllText(filePath, textEditor.Text); // Используем File.WriteAllText

MessageBox.Show("Сохранено успешно");

}

catch (Exception ex)

{

MessageBox.Show("Ошибка сохранения: " + ex.Message);

}

}

private void BackButton\_Click(object sender, RoutedEventArgs e)

{

textEditor.Undo();

}

private void ForwardButton\_Click(object sender, RoutedEventArgs e)

{

textEditor.Redo();

}

private void DeleteAll(object sender, RoutedEventArgs e)

{

textEditor.Clear();

}

private void RCB1\_TextChanged(object sender, TextChangedEventArgs e)

{

//Dispatcher.InvokeAsync(HighlightLastWord, DispatcherPriority.ContextIdle);

}

private void CopyButton\_Click(object sender, RoutedEventArgs e)

{

string selectedText = textEditor.SelectedText;

if (!string.IsNullOrEmpty(selectedText))

{

Clipboard.SetText(selectedText);

}

}

private void CutButton\_Click(object sender, RoutedEventArgs e)

{

if (!string.IsNullOrEmpty(textEditor.SelectedText))

{

Clipboard.SetText(textEditor.SelectedText);

textEditor.SelectedText = "";

}

}

private void PasteButton\_Click(object sender, RoutedEventArgs e)

{

if (Clipboard.ContainsText(TextDataFormat.Rtf))

{

try

{

string rtfText = Clipboard.GetText(TextDataFormat.Rtf);

textEditor.SelectedText = rtfText;

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при вставке RTF: " + ex.Message);

}

}

else if (Clipboard.ContainsText())

{

string clipboardText = Clipboard.GetText();

textEditor.SelectedText = clipboardText;

}

}

private void ExitApp(object sender, RoutedEventArgs e)

{

this.Close();

}

private void OpenFile\_Click(object sender, RoutedEventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Multiselect = false;

openFileDialog.Filter = "Текстовые файлы (\*.txt)|\*.txt|Все файлы (\*.\*)|\*.\*";

if (openFileDialog.ShowDialog() == true)

{

try

{

string filepath = openFileDialog.FileName;

string fileContent = File.ReadAllText(filepath);

textEditor.Text = "";

textEditor.Text += fileContent;

filePath = filepath;

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

}

}

private void ManualButton\_Click(object sender, RoutedEventArgs e)

{

TextWindow textWindow = new TextWindow("Справка");

textWindow.ShowDialog();

}

private void AboutProgrammButton\_Click(object sender, RoutedEventArgs e)

{

TextWindow textWindow = new TextWindow("О программе");

textWindow.ShowDialog();

}

private void TryToExit(object sender, CancelEventArgs e)

{

e.Cancel = true;

ExitWindow confirmationWindow = new ExitWindow();

if (confirmationWindow.ShowDialog() == true)

{

e.Cancel = false;

Application.Current.Shutdown();

}

}

private void StartButton\_Click(object sender, RoutedEventArgs e)

{

RCB2.Document.Blocks.Clear();

List<string> ToWrite = LeksAnalisation.Analyze(textEditor);

if (ToWrite != null)

foreach (var token in ToWrite)

RCB2.AppendText(token + "\n");

else

RCB2.AppendText("Ошибок нет\n");

}

}

}

internal static class LeksAnalisation

{

private static readonly HashSet<string> \_variables = new HashSet<string>();

public static int GetIndexAfterSecondWordRegex(string input)

{

Match match = Regex.Match(input, @"^(\S+\s+){2}");

if (!match.Success)

{

return -1; // Не найдено два слова.

}

// Возвращаем индекс, где заканчивается совпадение (то есть, после второго слова и разделителя).

return match.Length;

}

static List<int> GetStartIndicesOfElements(string input)

{

List<int> indices = new List<int>();

// Регулярное выражение для поиска:

// - слов (например, "Complex", "s", "new"),

// - чисел (например, "3", "0"),

// - отдельных символов (например, "=", "(", ",", ")", ";").

string pattern = @"[a-zA-Z\_]\w\*|\d+|[^\w\s]";

MatchCollection matches = Regex.Matches(input, pattern);

foreach (Match match in matches)

{

if (match.Success)

{

indices.Add(match.Index);

}

}

return indices;

}

static List<string> SplitTokens(string input)

{

List<string> tokens = new List<string>();

string pattern = @"(-?\d+(?:\.\d+)?(?:e[+-]?\d+)?)|([a-zA-Zа-яА-Я\_]\w\*)|([=(),])|\s+";

MatchCollection matches = Regex.Matches(input, pattern);

foreach (Match match in matches)

{

string value = match.Value.Trim();

if (!string.IsNullOrEmpty(value))

{

tokens.Add(value);

}

}

MessageBox.Show(tokens.Count().ToString());

return tokens;

}

static Dictionary<int, string> Words = new Dictionary<int, string>() { {0, "Complex" }, { 1, @"^@?[a-zA-Z\_]\w\*$" }, { 2,"=" },{3,"new" },{ 4,"Complex" },{5,"(" }, {6, @"^[+-]?(?:\d+\.?\d\*|\.\d+)(?:[eE][+-]?\d+)?f?$" }, {7,"," }, {8,"ffff" }, };

private static readonly HashSet<char> InvalidChars = new HashSet<char>

{

'@', '#', '!', '&', '\*', '[', ']', '{', '}', '~', '$', '%', '^', '`', '\\', '|', '<', '>', '?', ':', '/', '"',

};

private class TokenPattern

{

public string Type { get; }

public string Pattern { get; }

public string Description { get; }

public bool IsCritical { get; }

public TokenPattern(string type, string pattern, string description, bool isCritical = false)

{

Type = type;

Pattern = pattern;

Description = description;

IsCritical = isCritical;

}

}

private static readonly List<TokenPattern> ExpectedPatterns = new List<TokenPattern>

{

new TokenPattern("Complex1", "Complex", "первое 'Complex'", true),

new TokenPattern("Name", @"[a-zA-Z\_][a-zA-Z0-9\_]\*", "имя переменной"),

new TokenPattern("Assign", "=", "оператор '='"),

new TokenPattern("New", "new", "'new'", true),

new TokenPattern("Complex2", "Complex", "второе 'Complex'", true),

new TokenPattern("OpenParen", "(", "'('"),

new TokenPattern("Number1", @"[+-]?(\d+\.?\d\*|\.\d+)([eE][+-]?\d+)?[fFdDmM]?", "первое число"),

new TokenPattern("Comma", ",", "','"),

new TokenPattern("Number2", @"[+-]?(\d+\.?\d\*|\.\d+)([eE][+-]?\d+)?[fFdDmM]?", "второе число"),

new TokenPattern("CloseParen", ")", "')'"),

new TokenPattern("Semicolon", ";", "';'")

};

private static List<string> RegularTokens = new List<string>() { "Complex", @"^[a-zA-Z\_][a-zA-Z0-9\_]\*$", @"=", "new", "Complex", @"(", @"^-?\d+(\.\d+)?(f|e[-+]?\d+)?$", @",", @"^-?\d+(\.\d+)?(f|e[-+]?\d+)?$", @")", @";" };

private static Dictionary<string, string> TokenNameMap = new Dictionary<string, string>()

{

{ @"^[a-zA-Z\_][a-zA-Z0-9\_]\*$", "имя переменной" },

{ @"^-?\d+(\.\d+)?(f|e[-+]?\d+)?$", "число" },

{ "(", "(" }, // Добавлено для левой скобки

{ ")", ")" }, // Добавлено для правой скобки

{ ";", ";" } // Добавлено для точки с запятой

};

public static List<string> Analyze(TextEditor textEditor)

{

var results = new List<string>();

var document = textEditor.Document;

for (int i = 0; i < document.LineCount; i++)

{

int lineNumber = i + 1; // Номер текущей строки

bool hasErrorsInLine = false; // Флаг для отслеживания ошибок в текущей строке

List<bool> RegularTokensExistance = new List<bool>(new bool[RegularTokens.Count]);

(string cleaned, List<(int, int)> ranges, List<string> substrings) = CleanString(document.GetText(document.GetLineByNumber(i + 1)));

if (substrings.Count > 0)

{

hasErrorsInLine = true;

for (int h = 0; h < substrings.Count; h++)

{

results.Add($"Строка {lineNumber}: Обнаружен недопустимый токен: " + substrings[h] + " с " + ranges[h].Item1 + " по " + ranges[h].Item2 + "\n");

}

}

List<(string Token, int Index)> Smth = Tokenize(cleaned);

int tokenIndex = 0;

int expectedTokenIndex = 0;

while (tokenIndex < Smth.Count)

{

bool foundMatch = false;

for (int j = expectedTokenIndex; j < RegularTokens.Count; j++)

{

if (!RegularTokensExistance[j])

{

if (Smth[tokenIndex].Token == RegularTokens[j])

{

RegularTokensExistance[j] = true;

foundMatch = true;

expectedTokenIndex = j + 1;

tokenIndex++;

break;

}

else

{

try

{

if (Regex.IsMatch(Smth[tokenIndex].Token, RegularTokens[j]))

{

if (j == 1 && (Smth[tokenIndex].Token == "Complex" || Smth[tokenIndex].Token == "new"))

{

hasErrorsInLine = true;

results.Add($"Строка {lineNumber}: Ошибка: Нельзя использовать ключевое слово '" + Smth[tokenIndex].Token + "' в качестве имени переменной.\n");

foundMatch = true;

tokenIndex++;

break;

}

RegularTokensExistance[j] = true;

foundMatch = true;

expectedTokenIndex = j + 1;

tokenIndex++;

break;

}

}

catch (ArgumentException) { /\* Обработка исключения \*/ }

}

}

}

if (!foundMatch)

{

hasErrorsInLine = true;

results.Add($"Строка {lineNumber}: Лишний или неверный токен: " + Smth[tokenIndex].Token + " на позиции: " + Smth[tokenIndex].Index + "\n");

tokenIndex++;

}

}

for (int j = 0; j < RegularTokens.Count; j++)

{

if (!RegularTokensExistance[j])

{

hasErrorsInLine = true;

string tokenName = RegularTokens[j];

if (TokenNameMap.ContainsKey(RegularTokens[j]))

{

tokenName = TokenNameMap[RegularTokens[j]];

}

results.Add($"Строка {lineNumber}: Отсутствует обязательный токен: " + tokenName + "\n");

}

}

// Добавляем сообщение об отсутствии ошибок, если ошибок не было

if (!hasErrorsInLine)

{

results.Add($"Строка {lineNumber}: Ошибки не найдены.\n");

}

}

return results;

}

public static List<(string Token, int Index)> Tokenize(string text)

{

var tokens = new List<(string, int)>();

int currentIndex = 0;

while (currentIndex < text.Length)

{

// Пропускаем пробелы

if (char.IsWhiteSpace(text[currentIndex]))

{

currentIndex++;

continue;

}

// Ищем идентификатор (переменную)

if (char.IsLetter(text[currentIndex]) || text[currentIndex] == '\_')

{

int startIndex = currentIndex;

while (currentIndex < text.Length && (char.IsLetterOrDigit(text[currentIndex]) || text[currentIndex] == '\_'))

{

currentIndex++;

}

string identifier = text.Substring(startIndex, currentIndex - startIndex);

tokens.Add((identifier, startIndex));

continue;

}

// Ищем число

if (char.IsDigit(text[currentIndex]) || text[currentIndex] == '-')

{

int startIndex = currentIndex;

//Более сложная логика разбора числа (с учетом десятичной точки, экспоненты и т.д.)

while (currentIndex < text.Length && (char.IsDigit(text[currentIndex]) || text[currentIndex] == '.' || text[currentIndex] == 'e' || text[currentIndex] == 'E' || text[currentIndex] == '+' || text[currentIndex] == '-' || char.ToLower(text[currentIndex]) == 'f')) // Добавлено char.ToLower(text[currentIndex]) == 'f'

{

currentIndex++;

}

string number = text.Substring(startIndex, currentIndex - startIndex);

tokens.Add((number, startIndex));

continue;

}

// Если не идентификатор и не число, считаем это отдельным токеном (оператор, скобка и т.д.)

tokens.Add((text[currentIndex].ToString(), currentIndex));

currentIndex++;

}

return tokens;

}

public static (string cleanedString, List<(int startIndex, int endIndex)> removedRanges, List<string> removedSubstrings) CleanString(string input)

{

if (string.IsNullOrEmpty(input))

{

return (input, new List<(int, int)>(), new List<string>());

}

StringBuilder cleanedString = new StringBuilder();

List<(int startIndex, int endIndex)> removedRanges = new List<(int, int)>();

List<string> removedSubstrings = new List<string>();

int removalStart = -1;

for (int i = 0; i < input.Length; i++)

{

if (InvalidChars.Contains(input[i]))

{

if (removalStart == -1)

{

removalStart = i;

}

}

else

{

if (removalStart != -1)

{

removedRanges.Add((removalStart, i - 1));

removedSubstrings.Add(input.Substring(removalStart, i - removalStart)); // Извлекаем подстроку

removalStart = -1;

}

cleanedString.Append(input[i]);

}

}

if (removalStart != -1)

{

removedRanges.Add((removalStart, input.Length - 1));

removedSubstrings.Add(input.Substring(removalStart)); // Извлекаем подстроку

}

return (cleanedString.ToString(), removedRanges, removedSubstrings);

}

}

internal class Token

{

public string Text { get; }

public int StartIndex { get; }

public int EndIndex { get; }

public int Code { get; }

public string Category { get; }

public Token(string text, int startIndex, int endIndex)

{

Text = text;

StartIndex = startIndex;

EndIndex = endIndex;

Code = GetTokenCode(text);

Category = GetTokenCategory(text);

}

public string IndexInfo => StartIndex == EndIndex ? $"{StartIndex}" : $"с {StartIndex} по {EndIndex}";

private int GetTokenCode(string text)

{

switch (text)

{

case "Complex": return 1;

case "new": return 2;

case " ": return 3;

case "=": return 5;

case "(": return 7;

case ")": return 8;

case ",": return 9;

case ";": return 10;

default:

if (Regex.IsMatch(text, @"^[a-zA-Z\_][a-zA-Z0-9\_]\*$")) return 4; // Идентификатор

if (Regex.IsMatch(text, @"^[+-]?\d+(**\.**\d+)?([eE][+-]?\d+)?$")) return 6; // Число

return -1; // Неизвестная лексема

}

}

private string GetTokenCategory(string text)

{

switch (text)

{

case "Complex": return "Ключевое слово";

case "new": return "Ключевое слово";

case " ": return "Разделитель";

case "=": return "Оператор присваивания";

case "(": return "Ограничитель";

case ")": return "Ограничитель";

case ",": return "Разделитель";

case ";": return "Конец оператора";

default:

if (Regex.IsMatch(text, @"^[a-zA-Z\_][a-zA-Z0-9\_]\*$")) return "Идентификатор";

if (Regex.IsMatch(text, @"^[+-]?\d+(**\.**\d+)?([eE][+-]?\d+)?$")) return "Число";

return "Неизвестная лексема";

}

}

}

namespace Compiler

{

internal static class Painter

{

//public static List<Color> ColorsOfTheText = new List<Color>();

public static readonly Dictionary<string, Color> ColorBook = new Dictionary<string, Color>() {

{ "new", Colors.Blue },

{ "Complew", Colors.Blue },

};

}

}

public partial class TextWindow : Window

{

private Dictionary<string, string> TextOfRCB = new Dictionary<string, string>() { {"Справка", "1. Введение\r\n\r\nЭтот документ предоставляет информацию о том, как использовать программу-компилятор комплексных чисел, разработанную на языке C#. Программа позволяет вычислять значения выражений, содержащих комплексные числа и основные арифметические операции.\r\n\r\n2. Установка и запуск\r\n\r\nТребования: Для работы программы требуется установленная среда выполнения .NET.\r\nЗапуск: Запустите исполняемый файл программы.\r\n3. Пользовательский интерфейс\r\n\r\nОсновное окно программы содержит:\r\n\r\nПоле ввода: Текстовое поле для ввода выражения с комплексными числами.\r\nКнопка “Вычислить”: Запускает процесс компиляции и вычисления выражения.\r\nПоле вывода: Текстовое поле для отображения результата вычисления или сообщения об ошибке.\r\n4. Формат ввода выражений\r\nПрограмма поддерживает следующие форматы записи комплексных чисел:\r\nДекартова форма (a + bi):\r\na - действительная часть (действительное число).\r\nb - мнимая часть (действительное число).\r\ni - мнимая единица.\r\nПримеры: 2 + 3i, -1 - 0.5i, 0 + 1i, 5 + i (когда мнимая часть равна 1). 5-i\r\nПолярная форма (r \* e^(iθ) или r(cos(θ) + i\*sin(θ)) ):\r\nr - модуль (действительное число).\r\nθ - аргумент в радианах или градусах (действительное число). Укажите, что поддерживается.\r\nПримеры: 5 \* e^(i\*pi/2), 2(cos(pi/4) + i\*sin(pi/4)). (Требуется указать, как конкретно реализуется ввод полярной формы, если она есть.)\r\nПоддерживаемые операции:\r\nСложение: +\r\nВычитание: -\r\nУмножение: \*\r\nДеление: /\r\nПримеры выражений:\r\n2 + 3i + 1 - i\r\n(4 + 2i) \* (1 - i)\r\n(5 - i) / (2 + 2i)" },

{"О программе", "О программе:\r\n\r\nДанная программа представляет собой компилятор, разработанный на языке C#, для выполнения операций с комплексными числами. Он обеспечивает:\r\n\r\n\* \*\*Парсинг выражений:\*\* Анализ и разбор строковых выражений, содержащих комплексные числа в различных форматах.\r\n\* \*\*Вычисление:\*\* Поддержку основных арифметических операций (+, -, \*, /) над комплексными числами.\r\n\* \*\*Обработку ошибок:\*\* Обнаружение и сообщение об ошибках в входных выражениях.\r\n\* \*\*Вывод результатов:\*\* Отображение вычисленных комплексных чисел в удобном формате.\r\n\r\nПрограмма предназначена для помощи разработчикам и исследователям, работающим с комплексными числами." } };

public TextWindow(string text)

{

InitializeComponent();

this.Title = text;

BaseRCB.Document.Blocks.Clear();

BaseRCB.Document.Blocks.Add(new Paragraph(new Run(TextOfRCB[text])));

}

}

public partial class CreateWindow : Window

{

public string FileName { get; set; }

public CreateWindow()

{

InitializeComponent();

}

private void MegaButton\_Click(object sender, RoutedEventArgs e)

{

string fileName = TBFileName.Text.Trim();

// Проверка имени файла (допустимые символы, длина, и т.д.)

if (IsValidFileName(fileName))

{

FileName = fileName; // Сохраняем имя файла

DialogResult = true; // Закрываем окно с результатом "OK"

Close();

}

else

{

MessageBox.Show("Недопустимое имя файла.");

}

}

private bool IsValidFileName(string fileName)

{

// Пример: проверяем, что имя файла не пустое и не содержит недопустимых символов

if (string.IsNullOrEmpty(fileName)) return false;

if (fileName.IndexOfAny(Path.GetInvalidFileNameChars()) >= 0) return false;

return true;

}

}

using System.Windows;

[assembly: ThemeInfo(

ResourceDictionaryLocation.None, //where theme specific resource dictionaries are located

//(used if a resource is not found in the page,

// or application resource dictionaries)

ResourceDictionaryLocation.SourceAssembly //where the generic resource dictionary is located

//(used if a resource is not found in the page,

// app, or any theme specific resource dictionaries)

)]