Листинг программы

Parcer.cs

```
using System; using System.Collections.Generic; using System.Linq; using
WpfApp1.models;
public class Parser { private List _tokens; private int _currentIndex; public
List Errors { get; } = new List(); private HashSet _parameters = new HashSet();
public class ParseError
{

    public string Message { get; }
    public string Position { get; }

    public ParseError(string message, string position)
    {
        Message = message;
        Position = position;
    }

    public override string ToString() => $"{Message}\tПозиция: {Position}";
}

public Parser(List<Token> tokens)
{
    _tokens = tokens.Where(t => t.Code != 7 && t.Code != 8).ToList();
    _currentIndex = 0;
}

private Token CurrentToken => _currentIndex < _tokens.Count ?
_tokens[_currentIndex] : null;
private void MoveNext() => _currentIndex++;

private void AddError(string expected, Token actual)
{
    if (actual == null) return;
    Errors.Add(new ParseError($"{expected}, получено '{actual.Lexeme}'",
actual.Position));
}

public void Parse()
{
    var errorTokens = _tokens.Where(t => t.Code == 15).ToList();
    _currentIndex = 0;

    foreach (var token in errorTokens)
    {
        Errors.Add(new ParseError($"Неожиданный символ '{token.Lexeme}'",
token.Position));
    }

    ParseFunction();

    if (CurrentToken != null && CurrentToken.Code != 14)
    {
        AddError("Неожиданный символ после функции", CurrentToken);
    }
```

```
    End();
}

private void ParseFunction()
{
    if (CurrentToken == null) return;

    if (CurrentToken.Code == 1)
    {
        MoveNext();
    }
    else
    {
        bool looksLikeFunction = CurrentToken.Lexeme?.StartsWith("",
StringComparison.Ordinal) ?? false;
        if (CurrentToken.Lexeme.StartsWith("function", StringComparison.Ordinal)
&& CurrentToken.Code == 10)
        {
            AddError("Ожидался пробел после function", CurrentToken);
            MoveNext();
        }
        else if (looksLikeFunction && CurrentToken.Code != 15)
        {
            AddError("Ожидалось ключевое слово 'function'", CurrentToken);
            MoveNext();
        }

        while (CurrentToken.Code == 15)
        {
            MoveNext();
        }
    }

    while (CurrentToken.Code == 15)
    {
        MoveNext();
    }

    if (CurrentToken.Code == 10)
    {
        MoveNext();
    }
    else if(CurrentToken?.Code == 9 || CurrentToken?.Code == 10 ||
CurrentToken?.Code == 17) { }
    else
    {
        AddError("Ожидалось имя функции", CurrentToken);
    }

    while (CurrentToken.Code == 15)
    {
        MoveNext();
    }

    if (CurrentToken?.Code == 9)
    {
```

```
        MoveNext();
        ParseParameters();

        if (CurrentToken?.Code != 11)
        {
            AddError("Ожидалась ')'", CurrentToken);
        }
        else
        {
            MoveNext();
        }
    }
    else
    {
        AddError("Ожидалась '(' после имени функции", CurrentToken);
        ParseParameters();

        if (CurrentToken?.Code == 11)
        {
            MoveNext();
        }
    }

    while (CurrentToken.Code == 15)
    {
        MoveNext();
    }

    if (CurrentToken?.Code == 12)
    {
        MoveNext();
        ParseReturnStatement();

        if (CurrentToken == null || CurrentToken.Code != 13)
        {
            string errorPosition = "end";
            if (CurrentToken != null && CurrentToken.Code == 11)
            {
                AddError("Ожидалась '}'", CurrentToken);
                errorPosition = CurrentToken.Position;
            }
            AddError("Ожидалась '}'", new Token { Position = errorPosition });
        }
        else
        {
            MoveNext();
        }
    }
    else
    {
        if (CurrentToken?.Code != 12)
        {
            AddError("Ожидалось '{'", CurrentToken);
            ParseReturnStatement();

            if (CurrentToken == null || CurrentToken.Code != 13)
            {
                string errorPosition = "end";
```

```
                if (CurrentToken != null && CurrentToken.Code == 11)
                {
                    AddError("Ожидалась '}'", CurrentToken);
                    errorPosition = CurrentToken.Position;
                }
                AddError("Ожидалась '}'", new Token { Position = errorPosition
});
            }
            else
            {
                MoveNext();
            }
        }
        else
        {
            AddError("Ожидалось '{'", CurrentToken);
        }
    }
}

private void ParseParameters()
{
    _parameters.Clear();
    bool expectParam = true;
    int safetyCounter = 0;
    const int maxIterations = 100;

    while (CurrentToken != null &&
           CurrentToken.Code != 11 &&
           CurrentToken.Code != 12 &&
           safetyCounter++ < maxIterations)
    {
        if (CurrentToken.Code == 15)
        {
            MoveNext();
            continue;
        }

        if (expectParam)
        {
            if (CurrentToken.Code == 10)
            {
                if (CurrentToken.Lexeme.Length > 0 &&
char.IsDigit(CurrentToken.Lexeme[0]))
                {
                    AddError("Идентификатор параметра не может начинаться с
цифры", CurrentToken);
                }

                _parameters.Add(CurrentToken.Lexeme);
                MoveNext();
                expectParam = false;

                if (CurrentToken?.Code == 10)
                {
                    AddError("Ожидалась ',' между идентификаторами",
CurrentToken);
                    expectParam = true;
```

```
                }
                else if (CurrentToken?.Code != 17 &&
                         CurrentToken?.Code != 11 &&
                         CurrentToken?.Code != 12)
                {
                    expectParam = true;
                }
            }
            else if (CurrentToken.Code == 17)
            {
                if (_currentIndex > 0 && _tokens[_currentIndex - 1].Code == 17)
                {
                    AddError("Ожидался параметр после ','", CurrentToken);
                    MoveNext();
                }
                else if (_currentIndex == 0 ||
                        _tokens[_currentIndex - 1].Code == 9)
                {
                    AddError("Ожидался параметр перед ','", CurrentToken);
                    MoveNext();
                }
                else
                {
                    MoveNext();
                    expectParam = true;
                }
            }
            else if (CurrentToken.Code == 12)
            {
                if (_currentIndex > 0 && _tokens[_currentIndex - 1].Code == 17)
                {
                    AddError("Ожидался параметр после ','",
_tokens[_currentIndex - 1]);
                }
                break;
            }
            else
            {
                break;
            }
        }
        else
        {
            if (CurrentToken.Code == 17)
            {
                MoveNext();
                expectParam = true;
            }
            else if (CurrentToken.Code == 10)
            {
                AddError("Ожидалась ',' между идентификаторами", CurrentToken);
                expectParam = true;
            }
            else if (CurrentToken.Code != 11 && CurrentToken.Code != 12)
            {
                expectParam = true;
            }
            else
            {
```

```
                break;
            }
        }
    }

    if (safetyCounter >= maxIterations)
    {
        AddError("Ошибка разбора параметров (возможное зацикливание)",
CurrentToken);
    }

    if (CurrentToken?.Code == 11 &&
        _currentIndex > 0 &&
        _tokens[_currentIndex - 1].Code == 17)
    {
        AddError("Ожидался параметр после ','", _tokens[_currentIndex - 1]);
    }
    else if (CurrentToken?.Code == 12 &&
             _currentIndex > 0 &&
             _tokens[_currentIndex - 1].Code == 17)
    {
        AddError("Ожидался параметр после ','", _tokens[_currentIndex - 1]);
    }
}

private void ParseReturnStatement()
{
    if (CurrentToken == null) return;

    if (CurrentToken.Code == 6)
    {
        MoveNext();
    }
    else
    {
        bool looksLikeReturn = CurrentToken.Lexeme?.StartsWith("",
StringComparison.Ordinal) ?? false;
        if (CurrentToken.Lexeme.StartsWith("return", StringComparison.Ordinal)
&& CurrentToken.Code == 10)
        {
            AddError("Ожидался пробел после return", CurrentToken);
        }
        else if (looksLikeReturn && CurrentToken.Code != 15)
        {
            AddError("Ожидалось ключевое слово 'return'", CurrentToken);
        }

        MoveNext();
    }

    ParseExpression();
}

private void ParseExpression()
{
    if (CurrentToken != null && (CurrentToken.Code == 16 || CurrentToken.Code ==
18 || CurrentToken.Code == 19 || CurrentToken.Code == 20))
```

```
    {
        AddError("Выражение не может начинаться с оператора", CurrentToken);
        MoveNext();
    }
    while (CurrentToken != null && CurrentToken.Code != 13 && CurrentToken.Code
!= 14)
    {
        if (CurrentToken.Code == 15)
        {
            MoveNext();
            continue;
        }

        switch (CurrentToken.Code)
        {
            case 9:
                MoveNext();
                ParseExpression();
                if (CurrentToken == null || CurrentToken.Code != 11)
                {
                    AddError("Ожидалась ')'", CurrentToken ?? new Token {
Position = "end" });
                    return;
                }
                MoveNext();
                break;

            case 10:
            case 21:
                var prevToken = _currentIndex > 0 ? _tokens[_currentIndex - 1] :
null;
                MoveNext();

                if (CurrentToken != null && CurrentToken.Code != 15 &&
                    (CurrentToken.Code == 10 || CurrentToken.Code == 21 ||
CurrentToken.Code == 9))
                {
                    AddError("Ожидался оператор между выражениями",
CurrentToken);
                }
                break;

            case 11:
                if (!HasMatchingOpeningBracket())
                {
                    AddError("Лишняя ')'", CurrentToken);
                    MoveNext();
                    return;
                }
                return;

            case 16:
            case 18:
            case 19:
            case 20:
                int currentOp = CurrentToken.Code;
                MoveNext();
```

```
                if (CurrentToken != null &&
                    (CurrentToken.Code == 16 || CurrentToken.Code == 18 ||
                     CurrentToken.Code == 19 || CurrentToken.Code == 20))
                {
                    AddError("Ожидался идентификатор между операторами",
CurrentToken);
                    continue;
                }

                if (CurrentToken != null && CurrentToken.Code != 15 &&
                    CurrentToken.Code != 10 && CurrentToken.Code != 21 &&
CurrentToken.Code != 9)
                {
                    AddError("Ожидалось выражение после оператора",
CurrentToken);
                }
                break;

            default:
                AddError("Ожидалась часть выражения", CurrentToken);
                MoveNext();
                break;
        }
    }
}

private bool HasMatchingOpeningBracket()
{
    int bracketCount = 1;
    for (int i = _currentIndex - 1; i >= 0; i--)
    {
        if (_tokens[i].Code == 11)
            bracketCount++;
        else if (_tokens[i].Code == 9)
            bracketCount--;

        if (bracketCount == 0)
            return true;
    }
    return false;
}

private void End()
{
    if (CurrentToken == null || CurrentToken.Code != 14)
    {
        Token endToken = new Token
        {
            Code = 0,
            Lexeme = "конец функции",
            Position = CurrentToken != null ? CurrentToken.Position : "end"
        };
        AddError("Ожидался ';' после объявления функции", endToken);
    }
    else
    {
        MoveNext();
    }
```

```
}

}
```

MainWindow.cs

```
using ICSharpCode.AvalonEdit;
using ICSharpCode.AvalonEdit.Highlighting;
using Microsoft.Win32;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text.RegularExpressions;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Media;
using WpfApp1.models;

namespace WpfApp1
{
    public partial class MainWindow : Window
    {
        private string currentFilePath;
        private readonly FileManager fileManager;
        private readonly EditManager editManager;
        private readonly HelpManager helpManager;

        private double _fontSize = 14;

        public double FontSize
        {
            get { return _fontSize; }
            set
            {
                if (_fontSize != value)
                {
                    _fontSize = value;
                    // Обновляем размер шрифта для всех элементов
                    InputTextEditor.FontSize = value;
                    LexerOutputRichTextBox.FontSize = value;
                    ParserOutputRichTextBox.FontSize = value;
                    OnPropertyChanged(nameof(FontSize)); // Уведомляем об
изменении
                }
            }
        }

        // Реализация INotifyPropertyChanged для уведомлений об изменении
свойств
        public event PropertyChangedEventHandler PropertyChanged;

        protected virtual void OnPropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
```

```
        public MainWindow()
        {
            InitializeComponent();
            DataContext = this;

            InputTextEditor.SyntaxHighlighting =
HighlightingManager.Instance.GetDefinition("C#");

            fileManager = new FileManager(InputTextEditor, FileNameTextBlock);
            editManager = new EditManager(InputTextEditor);
            helpManager = new HelpManager();
        }



        private void SaveAs_Click(object sender, RoutedEventArgs e)
        {
            fileManager.SaveAs(ref currentFilePath);
        }

        private void Redo_Click(object sender, RoutedEventArgs e)
        {
            editManager.Redo();
        }

        private void Undo_Click(object sender, RoutedEventArgs e)
        {
            editManager.Undo();
        }

        private void OpenFile_Click(object sender, RoutedEventArgs e)
        {
            fileManager.OpenFile(ref currentFilePath);
        }

        private void CreateFile_Click(object sender, RoutedEventArgs e)
        {
            fileManager.CreateFile(ref currentFilePath);
        }

        private void SaveFile_Click(object sender, RoutedEventArgs e)
        {
            fileManager.SaveFile(ref currentFilePath);
        }

        private void Cut_Click(object sender, RoutedEventArgs e)
        {
            editManager.Cut();
        }

        private void Copy_Click(object sender, RoutedEventArgs e)
        {
            editManager.Copy();
        }

        private void Paste_Click(object sender, RoutedEventArgs e)
        {
            editManager.Paste();
```

```
        }

        private void Delete_Click(object sender, RoutedEventArgs e)
        {
            editManager.Delete();
        }

        private void SelectAll_Click(object sender, RoutedEventArgs e)
        {
            editManager.SelectAll();
        }

        private void Help_Click(object sender, RoutedEventArgs e)
        {
            helpManager.ShowHelp();
        }

        private void About_Click(object sender, RoutedEventArgs e)
        {
            helpManager.ShowAbout();
        }

        private void Exit_Click(object sender, RoutedEventArgs e)
        {
            Application.Current.Shutdown();
        }

        private void Window_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
        {
            if (!string.IsNullOrWhiteSpace(InputTextEditor.Text))
            {
                var result = MessageBox.Show("Вы хотите сохранить изменения?",
"Подтверждение", MessageBoxButton.YesNoCancel, MessageBoxImage.Warning);

                if (result == MessageBoxResult.Yes)
                {
                    fileManager.SaveAs(ref currentFilePath);
                }
                else if (result == MessageBoxResult.Cancel)
                {
                    e.Cancel = true;
                }
            }
        }

        private void AnalyzeButton_Click(object sender, RoutedEventArgs e)
        {
            // Очищаем RichTextBox
            LexerOutputRichTextBox.Document.Blocks.Clear();
            ParserOutputRichTextBox.Document.Blocks.Clear();

            string input = InputTextEditor.Text;

            // Лексический анализ
            Lexer lexer = new Lexer();
            List<Token> tokens = lexer.Analyze(input);

            // Вывод токенов в виде таблицы
```

```
            var lexerTable = new Table();
            lexerTable.Columns.Add(new TableColumn { Width = new GridLength(1,
GridUnitType.Star) });
            lexerTable.Columns.Add(new TableColumn { Width = new GridLength(1,
GridUnitType.Star) });
            lexerTable.Columns.Add(new TableColumn { Width = new GridLength(1,
GridUnitType.Star) });

            var lexerHeaderRowGroup = new TableRowGroup();
            var lexerHeaderRow = new TableRow { Background = Brushes.LightGray
};
            lexerHeaderRow.Cells.Add(new TableCell(new Paragraph(new Run("Тип"))
{ FontWeight = FontWeights.Bold }));
            lexerHeaderRow.Cells.Add(new TableCell(new Paragraph(new
Run("Лексема")) { FontWeight = FontWeights.Bold }));
            lexerHeaderRow.Cells.Add(new TableCell(new Paragraph(new
Run("Позиция")) { FontWeight = FontWeights.Bold }));
            lexerHeaderRowGroup.Rows.Add(lexerHeaderRow);
            lexerTable.RowGroups.Add(lexerHeaderRowGroup);

            var lexerDataRowGroup = new TableRowGroup();
            foreach (var token in tokens)
            {
                var row = new TableRow();
                row.Cells.Add(new TableCell(new Paragraph(new
Run(token.Type))));
                row.Cells.Add(new TableCell(new Paragraph(new
Run(token.Lexeme))));
                row.Cells.Add(new TableCell(new Paragraph(new
Run(token.Position))));
                lexerDataRowGroup.Rows.Add(row);
            }
            lexerTable.RowGroups.Add(lexerDataRowGroup);
            LexerOutputRichTextBox.Document.Blocks.Add(lexerTable);

            // Синтаксический анализ
            Parser parser = new Parser(tokens);
            parser.Parse();

            // Сортируем ошибки: сначала по числовой позиции, затем "end" в
конце
            var sortedErrors = parser.Errors.OrderBy(error =>
            {
                if (error.Position == "end")
                    return int.MaxValue;
                return GetPositionValue(error.Position);
            }).ThenBy(error => error.Position).ToList();

            // Вывод ошибок парсера
            var parserTable = new Table();
            parserTable.Columns.Add(new TableColumn { Width = new GridLength(2,
GridUnitType.Star) });
            parserTable.Columns.Add(new TableColumn { Width = new GridLength(1,
GridUnitType.Star) });

            var parserHeaderRowGroup = new TableRowGroup();
            var parserHeaderRow = new TableRow { Background = Brushes.LightGray
};
```

```
            // Добавляем строку с количеством ошибок
            var errorCountParagraph = new Paragraph();
            errorCountParagraph.Inlines.Add(new Run($"Ошибки синтаксического
анализа (всего: {sortedErrors.Count})")
            {
                FontWeight = FontWeights.Bold,
                Foreground = sortedErrors.Count > 0 ? Brushes.Red :
Brushes.Green
            });

            parserHeaderRow.Cells.Add(new TableCell(errorCountParagraph));
            parserHeaderRow.Cells.Add(new TableCell(new Paragraph(new
Run("Позиция")) { FontWeight = FontWeights.Bold }));
            parserHeaderRowGroup.Rows.Add(parserHeaderRow);
            parserTable.RowGroups.Add(parserHeaderRowGroup);

            var parserDataRowGroup = new TableRowGroup();
            if (sortedErrors.Count == 0)
            {
                var row = new TableRow();
                row.Cells.Add(new TableCell(new Paragraph(new Run("✓
Синтаксический анализ завершен успешно"))
                {
                    Foreground = Brushes.Green,
                    FontWeight = FontWeights.Bold
                }));
                row.Cells.Add(new TableCell(new Paragraph(new Run(""))));
                parserDataRowGroup.Rows.Add(row);
            }
            else
            {
                foreach (var error in sortedErrors)
                {
                    var row = new TableRow();
                    row.Cells.Add(new TableCell(new Paragraph(new
Run(error.Message)) { Foreground = Brushes.Red }));
                    row.Cells.Add(new TableCell(new Paragraph(new
Run(error.Position))));
                    parserDataRowGroup.Rows.Add(row);
                }
            }
            parserTable.RowGroups.Add(parserDataRowGroup);
            ParserOutputRichTextBox.Document.Blocks.Add(parserTable);
        }

        private int GetPositionValue(string position)
        {
            if (string.IsNullOrEmpty(position) || position == "end")
                return int.MaxValue;

            // Обрабатываем позиции вида "10" или "12-16"
            if (position.Contains("-"))
            {
                var parts = position.Split('-');
                if (parts.Length > 0 && int.TryParse(parts[0], out int start))
                    return start;
            }
            else if (int.TryParse(position, out int singlePos))
            {
```

```
                return singlePos;
            }

            return int.MaxValue - 1; // Нечисловые позиции перед "end"
        }
    }
}
```

MainWindow.xaml

```
<Window x:Class="WpfApp1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:avalonEdit="http://icsharpcode.net/sharpdevelop/avalonedit"
        Title="Моя работа" Height="450" Width="800" Background="#FFF0F4F8"
        Closing="Window_Closing">

    <DockPanel>
        <!-- Меню -->
        <Menu DockPanel.Dock="Top" Background="#FF007ACC" Foreground="White">
            <MenuItem Header="Файл">
                <MenuItem Header="Создать" Click="CreateFile_Click"
Foreground="Black"/>
                <MenuItem Header="Открыть" Click="OpenFile_Click"
Foreground="Black"/>
                <MenuItem Header="Сохранить" Click="SaveFile_Click"
Foreground="Black"/>
                <MenuItem Header="Сохранить как" Click="SaveAs_Click"
Foreground="Black"/>
                <Separator />
                <MenuItem Header="Выход" Click="Exit_Click" Foreground="Black"/>
            </MenuItem>
            <MenuItem Header="Правка">
                <MenuItem Header="Отменить" Click="Undo_Click"
Foreground="Black"/>
                <MenuItem Header="Повторить" Click="Redo_Click"
Foreground="Black"/>
                <Separator />
                <MenuItem Header="Вырезать" Click="Cut_Click"
Foreground="Black"/>
                <MenuItem Header="Копировать" Click="Copy_Click"
Foreground="Black"/>
                <MenuItem Header="Вставить" Click="Paste_Click"
Foreground="Black"/>
                <MenuItem Header="Удалить" Click="Delete_Click"
Foreground="Black"/>
                <MenuItem Header="Выделить все" Click="SelectAll_Click"
Foreground="Black"/>
            </MenuItem>
            <MenuItem Header="Текст">
                <MenuItem Header="Постановка задачи" Foreground="Black"/>
                <MenuItem Header="Грамматика" Foreground="Black"/>
                <MenuItem Header="Классификация грамматики" Foreground="Black"/>
                <MenuItem Header="Метод анализа" Foreground="Black"/>
                <MenuItem Header="Диагностика и нейтрализация ошибок"
Foreground="Black"/>
                <MenuItem Header="Тестовый пример" Foreground="Black"/>
                <MenuItem Header="Список литературы" Foreground="Black"/>
```

```
                <MenuItem Header="Исходный код программы" Foreground="Black"/>
            </MenuItem>
            <MenuItem Header="Пуск">
                <MenuItem Header="Запуск синтаксического анализатора"
Click="AnalyzeButton_Click" Foreground="Black"/>
            </MenuItem>
            <MenuItem Header="Справка">
                <MenuItem Header="Вызов справки" Click="Help_Click"
Foreground="Black"/>
                <MenuItem Header="О программе" Click="About_Click"
Foreground="Black"/>
            </MenuItem>
        </Menu>

        <!-- Панель инструментов -->
        <ToolBarTray DockPanel.Dock="Top" Background="LightGray">
            <ToolBar Band="1" BandIndex="1" Height="40">
                <Button ToolTip="Создать" Click="CreateFile_Click">
                    <Image Source="images/create.png" Width="20" Height="20" />
                </Button>
                <Button ToolTip="Открыть" Click="OpenFile_Click">
                    <Image Source="images/open.png" Width="20" Height="20" />
                </Button>
                <Button ToolTip="Сохранить" Click="SaveFile_Click">
                    <Image Source="images/save.png" Width="20" Height="20" />
                </Button>
                <Separator />
                <Button ToolTip="Отмена" Click="Undo_Click">
                    <Image Source="images/undo.png" Width="20" Height="20" />
                </Button>
                <Button ToolTip="Повтор" Click="Redo_Click">
                    <Image Source="images/redo.png" Width="20" Height="20" />
                </Button>
                <Separator />
                <Button ToolTip="Копировать" Click="Copy_Click">
                    <Image Source="images/copyfile.png" Width="20" Height="20"
/>
                </Button>
                <Button ToolTip="Вырезать" Click="Cut_Click">
                    <Image Source="images/cut.png" Width="20" Height="20" />
                </Button>
                <Button ToolTip="Вставить" Click="Paste_Click">
                    <Image Source="images/paste.png" Width="16" Height="16" />
                </Button>
                <Separator />
                <Button ToolTip="Запуск синтаксического анализатора"
Click="AnalyzeButton_Click">
                    <Image Source="images/start.png" Width="16" Height="16" />
                </Button>
                <Button ToolTip="Справка" Click="Help_Click">
                    <Image Source="images/help.png" Width="16" Height="16" />
                </Button>
                <Button ToolTip="О программе" Click="About_Click">
                    <Image Source="images/about.png" Width="16" Height="16" />
                </Button>
                <Separator />

                <ComboBox x:Name="FontSizeComboBox" Width="80"
VerticalAlignment="Center"
```

```
                SelectedValue="{Binding FontSize, Mode=TwoWay}"
                SelectedValuePath="Content">
                    <ComboBoxItem>12</ComboBoxItem>
                    <ComboBoxItem>14</ComboBoxItem>
                    <ComboBoxItem>16</ComboBoxItem>
                    <ComboBoxItem>18</ComboBoxItem>
                </ComboBox>
                <TextBlock x:Name="FileNameTextBlock" FontWeight="Normal"
FontSize="14" Width="Auto" Background="LightGray" VerticalAlignment="Center"
Margin="10,0,0,0"/>
            </ToolBar>
        </ToolBarTray>

        <!-- Основное содержимое -->
        <Grid Margin="5">
            <Grid.RowDefinitions>
                <RowDefinition Height="*"/>
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>

            <!-- AvalonEdit для ввода -->
            <avalonEdit:TextEditor
                Grid.Row="0" Grid.Column="1"
                Name="InputTextEditor"
                FontSize="{Binding FontSize}"
                ShowLineNumbers="True"
                FontFamily="Consolas"
                SyntaxHighlighting="C#"
                VerticalScrollBarVisibility="Auto"
                 />

            <!-- TabControl для вывода -->
            <TabControl Grid.Row="1" Grid.Column="1" x:Name="OutputTabControl">
                <TabItem Header="Лексер">
                    <RichTextBox x:Name="LexerOutputRichTextBox"
IsReadOnly="True" VerticalScrollBarVisibility="Auto" />
                </TabItem>
                <TabItem Header="Парсер">
                    <RichTextBox x:Name="ParserOutputRichTextBox"
IsReadOnly="True" VerticalScrollBarVisibility="Auto" />
                </TabItem>
            </TabControl>
        </Grid>
    </DockPanel>
</Window>
```