Приложение В

Листинг программы

Parcer.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using kursach;

public class Parser
{
    private readonly List<Token> _tokens;
    private int _position;
    private List<string> _errors = new();
    private HashSet<string> _parameters = new();

    public Parser(List<Token> tokens)
    {
        _tokens = tokens;
    }

    public List<string> Parse()
    {
        _errors.Clear();
        _position = 0;
        _parameters.Clear();

        if (!Match(TokenType.Идентификатор, out Token identifier))
        {
            AddError("Ожидался идентификатор в начале", Current());
        }

        if (!Match(TokenType.Присваивание))
        {
            AddError($"Ожидался оператор '=' после идентификатора
'{identifier.Value}'", Current());
        }

        if (!Match(TokenType.ОткрывающаяСкобка))
        {
            AddError("Ожидалась открывающая скобка '('", Current());
        }
```

```
        if (!MatchArguments()) return _errors;

        if (!Match(TokenType.Стрелка))
        {
            AddError("Ожидался оператор '->' после списка аргументов",
Current());
        }

        MatchExpression();

        if (!Match(TokenType.ТочкаСЗапятой))
        {
            AddError("Ожидался символ ';' в конце выражения", Current());
        }

        if (_position < _tokens.Count)
        {
            AddError("Лишние токены после конца выражения", Current());
        }

        return _errors;
    }

    private bool MatchArguments()
    {
        if (!Match(TokenType.Идентификатор, out Token id))
        {
            AddError("Ожидался хотя бы один идентификатор в списке
аргументов", Current());
            return false;
        }
        _parameters.Add(id.Value);

        while (Match(TokenType.Запятая))
        {
            if (!Match(TokenType.Идентификатор, out Token next))
            {
                AddError("Ожидался идентификатор после запятой",
Current());
                return false;
            }
```

```
            _parameters.Add(next.Value);
        }

        if (!Match(TokenType.ЗакрывающаяСкобка))
        {
            AddError("Ожидалась закрывающая скобка ')' после аргументов",
Current());
            return false;
        }
        return true;
    }

    private void MatchExpression()
    {
        MatchTerm();
        while (Match(TokenType.Оператор))
        {
            MatchTerm();
        }
    }

    private void MatchTerm()
    {
        if (Match(TokenType.Идентификатор, out Token id))
        {
            if (!_parameters.Contains(id.Value))
            {
                AddError($"Идентификатор  '{id.Value}'  не  объявлен  в
списке аргументов", id);
            }
        }
        else if (Match(TokenType.ОткрывающаяСкобка))
        {
            MatchExpression();
            if (!Match(TokenType.ЗакрывающаяСкобка))
            {
                AddError("Ожидалась  закрывающая  скобка  в  факторе",
Current());
            }
        }
        else
        {
```

```
                AddError("Ожидался идентификатор или выражение в скобках, но
найдено", Current());
                _position++;
            }
        }

        private bool Match(TokenType type) => Match(type, out _);

        private bool Match(TokenType type, out Token token)
        {
            if (_position < _tokens.Count && _tokens[_position].Type == type)
            {
                token = _tokens[_position++];
                return true;
            }
            token = null;
            return false;
        }

        private Token Current() => _position < _tokens.Count ?
_tokens[_position] : new Token(TokenType.Неизвестно, "EOF", _position);

        private void AddError(string message, Token token)
        {
            _errors.Add($"{message}: {token}");
        }
    }
```

Lexer.cs

```
using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;

namespace kursach
{
    public class Lexer
    {
        private readonly string _input;
        private int _position;

        private static readonly Regex IdentifierRegex = new Regex("^[a-
zA-Z_][a-zA-Z0-9_]*", RegexOptions.Compiled);

        public Lexer(string input)
        {
            _input = input;
        }
```

```
            public List<Token> Tokenize()
            {
                List<Token> tokens = new List<Token>();

                while (_position < _input.Length)
                {
                    char current = _input[_position];
                    int start = _position;

                    if (char.IsWhiteSpace(current))
                    {
                        _position++;
                        continue;
                    }
                    else if (current == '=')
                    {
                        tokens.Add(new Token(TokenType.Присваивание, "=",
_position++));
                    }
                    else if (current == '(')
                    {
                        tokens.Add(new Token(TokenType.ОткрывающаяСкобка,
"(", _position++));
                    }
                    else if (current == ')')
                    {
                        tokens.Add(new Token(TokenType.ЗакрывающаяСкобка,
")", _position++));
                    }
                    else if (current == ',')
                    {
                        tokens.Add(new Token(TokenType.Запятая, ",",
_position++));
                    }
                    else if (current == '-' && Peek() == '>')
                    {
                        tokens.Add(new Token(TokenType.Стрелка, "->",
_position));
                        _position += 2;
                    }
                    else if (current == '+' || current == '-' || current ==
'*' || current == '/')
                    {
                        tokens.Add(new Token(TokenType.Оператор,
current.ToString(), _position++));
                    }
                    else if (current == ';')
                    {
                        tokens.Add(new Token(TokenType.ТочкаСЗапятой, ";",
_position++));
                    }
                    else if (char.IsLetter(current) || current == '_')
                    {
                        var match =
IdentifierRegex.Match(_input.Substring(_position));
                        if (match.Success)
                        {
                            string value = match.Value;
                            tokens.Add(new Token(TokenType.Идентификатор,
value, _position));
                            _position += value.Length;
                        }
                        else
```

```
                        {
                            tokens.Add(new Token(TokenType.Неизвестно,
current.ToString(), _position++));
                        }
                    }
                    else
                    {
                        tokens.Add(new Token(TokenType.Неизвестно,
current.ToString(), _position++));
                    }
                }

                return tokens;
            }

            private char Peek() => _position + 1 < _input.Length ?
_input[_position + 1] : '\0';
        }
    }
```

Token.cs

```
namespace kursach;

public enum TokenType
{
    Идентификатор,
    Присваивание,
    ОткрывающаяСкобка,
    ЗакрывающаяСкобка,
    Запятая,
    Стрелка,
    Оператор,
    ТочкаСЗапятой,
    Неизвестно
}

public class Token
{
    public TokenType Type { get; set; }
    public string Value { get; set; }
    public int Position { get; set; }

    public Token(TokenType type, string value, int position)
    {
        Type = type;
        Value = value;
        Position = position;
    }

    public override string ToString() => $"{Type}('{Value}') at
{Position}";
}
```