ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра автоматизированных систем управления

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ

**«***Логические выражения языка С++***»**

*дисциплина:* ***«****Теория формальных языков и компиляторов****»***

Выполнил:
Студент гр. *АВТ-912*, *АВТФ*

*Мазуров А. В.*

«___» _____ 20__г.

_____
(подпись)

Проверил:
*д.т.н., профессор*

*Шорников Ю. В.*

«___» _____ 20__ г.

_____
(подпись)

Новосибирск 2022

**СОДЕРЖАНИЕ**

# 1 ПОСТАНОВКА ЗАДАЧИ

Выполнить программную реализацию алгоритма синтаксического анализа логических выражений языка C++.

Логическое выражение в программировании — конструкция языка программирования, результатом вычисления которой является «истина» или «ложь».

Над логическими выражениями и переменными возможны операции, результатом которых так же являются «истина» и «ложь»:

- дизъюнкция (или, ||);
- конъюнкция (и, &&);
- отрицание (не, !);
- исключающее или (^);
- эквиваленция (==);
- не эквиваленция (! =).

В рамках выполнения курсовой работы предлагается ограничится двумя логическими операторами: дизъюнкцией (или, ||) и конъюнкцией (и, &&)

Таким образом, логическое выражение будет иметь следующий синтаксис:

$$id1 \ < \text{Оператор} > \ id2 \ \{ < \text{Оператор} > \ id3 \ \};$$

где id1, id2, id3 – идентификаторы, начинающиеся с буквы и состоящий из букв и(или) цифр,

<Оператор> - логический оператор, соответствующий дизъюнкции (или, ||) и конъюнкции (и, &&).

## 2 ПОРОЖДАЮЩАЯ ГРАММАТИКА

Порождающая грамматика логических выражений C++ имеет следующий вид:

$G[< \text{ЛВ} >] = \{V_T, V_N, < \text{ЛВ} >, P\}$

где

Множество правил вывода $P$:

1. $< \text{ЛВ} > \rightarrow < \text{ИД} > < \text{ЛО} > < \text{ИД} > \{< \text{ЛО} > < \text{ИД} >\}'; '$
2. $< \text{ИД} > \rightarrow < \text{Б} > \{ < \text{Б} > | < \text{Ц} > \}$
3. $< \text{ЛО} > \rightarrow$ "||" | "&&"
4. $< \text{Ц} > \rightarrow '0' | '1' | \ldots | '9'$
5. $< \text{Б} > \rightarrow 'a' | 'b' | \ldots | 'z' | 'A' | 'B' | \ldots | 'Z'$

Начальный нетерминал:

$Z = < \text{ЛВ} >$

Множество терминальных символов:

$V_T = \{ 'a', 'b', \ldots, 'z', 'A', 'B', \ldots, 'Z', '0', '1', \ldots, '9', ';', "||", "\&\&" \}$

Множество нетерминальных символов:

$V_N = \{ < \text{ЛВ} >, < \text{ЛО} >, < \text{Б} >, < \text{Ц} > \}$

Условные обозначения:

- ЛВ – логическое выражение;
- ИД – идентификатор;
- ЛО – логический оператор;
- Б – буква;
- Ц – цифра.

## 3 КЛАССИФИКАЦИЯ ГРАММАТИКИ

Грамматика G[Z] по классификации Хомского относится к контекстно-свободной и имеет вид:

$$A \to \alpha, \text{где } A \in V_n, \alpha \in V^* \text{ [1].}$$

В левой части допускаются только нетерминальные символы, а в правой в части могут присутствовать символы как терминального, так и нетерминального словарей.

Диаграмма состояний сканера лексем представлена на Рисунке 1:

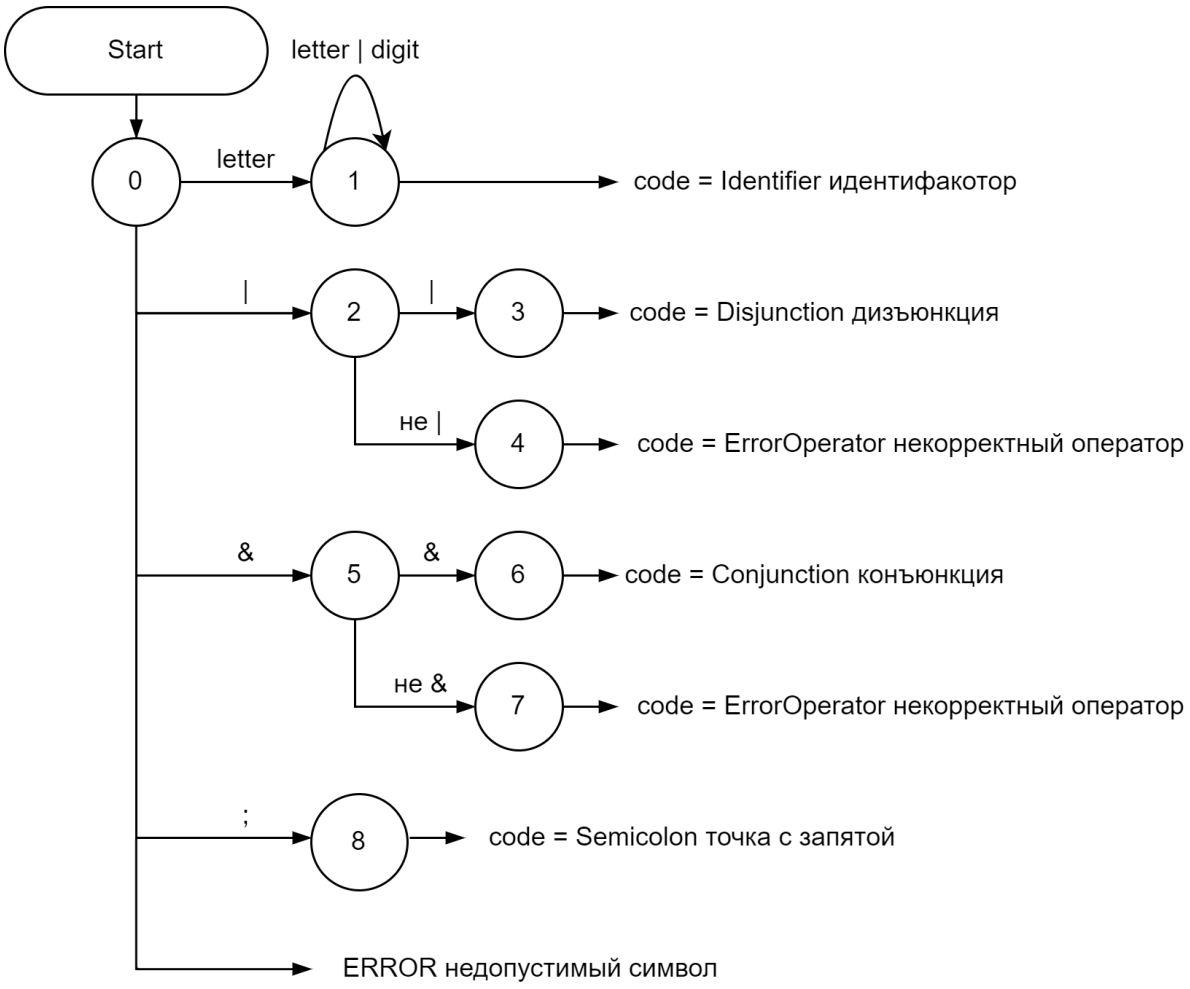


Рисунок 1 – диаграмма состояний сканера

# 4 МЕТОД АНАЛИЗА

Для грамматики G[Z] был выбран метод рекурсивного спуска, так как он подходит для контекстно-свободных грамматик.

Основная идея метода рекурсивного спуска заключается в следующем: каждому нетерминальному символу грамматики ставится в соответствие своя функция, процедура или другая программная единица, задача которой – начиная с указанного места исходной цепочки найти подцепочку, которая выводится из этого нетерминала. Тело каждой такой функции задаётся в соответствии с правилами вывода, соответствующего нетерминала: терминалы из правой части распознаются самой функцией, а нетерминалы соответствуют вызовам функций. Функции могут вызывать сами себя [1].

Для решения данной задачи выбран язык программирования C#.

# 5 ДИАГНОСТИКА И НЕЙТРАЛИЗАЦИЯ ОШИБОК

Для данной грамматики производится диагностика и нейтрализация ошибок с их возможным исправлением. Нейтрализация ошибок осуществляется по методу Айронса: спускаясь по синтаксическому дереву без возврата по контексту, при обнаружении тупиковой ситуации отбрасываются те литеры (символы), которые привели в тупиковую ситуацию. На их место ставится типовое значение с целью дальнейшего корректного разбора.

# 6 ТЕСТИРОВАНИЕ ПРОГРАММЫ

Результаты тестирования программы приведены далее на Рисунках 2–7:

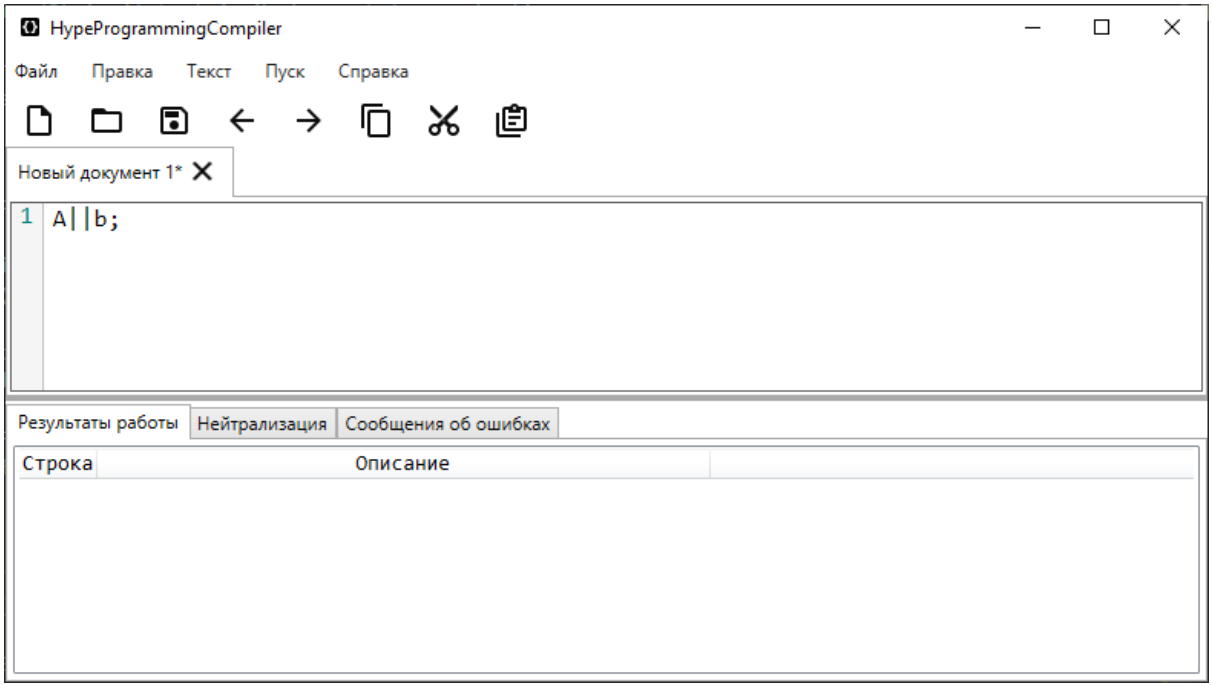


Рисунок 2 – ввод корректного логического выражения

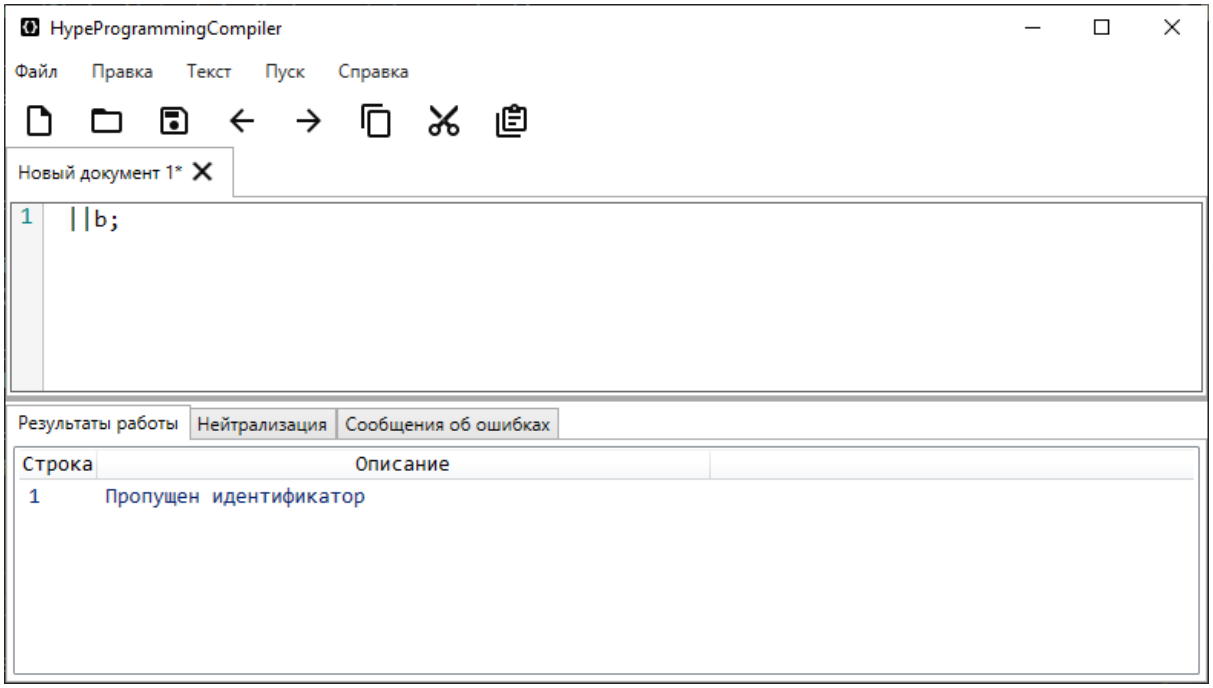


Рисунок 3 – пропуск идентификатора в логическом выражении

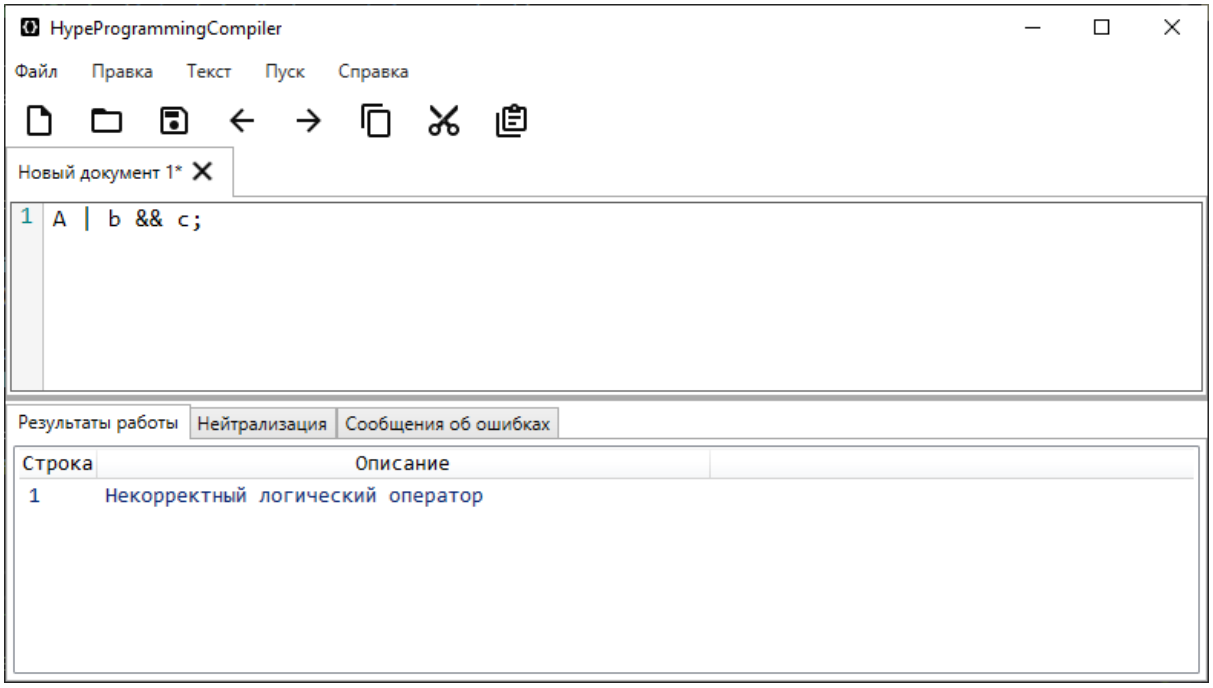


Рисунок 4 – ввод некорректного логического оператора

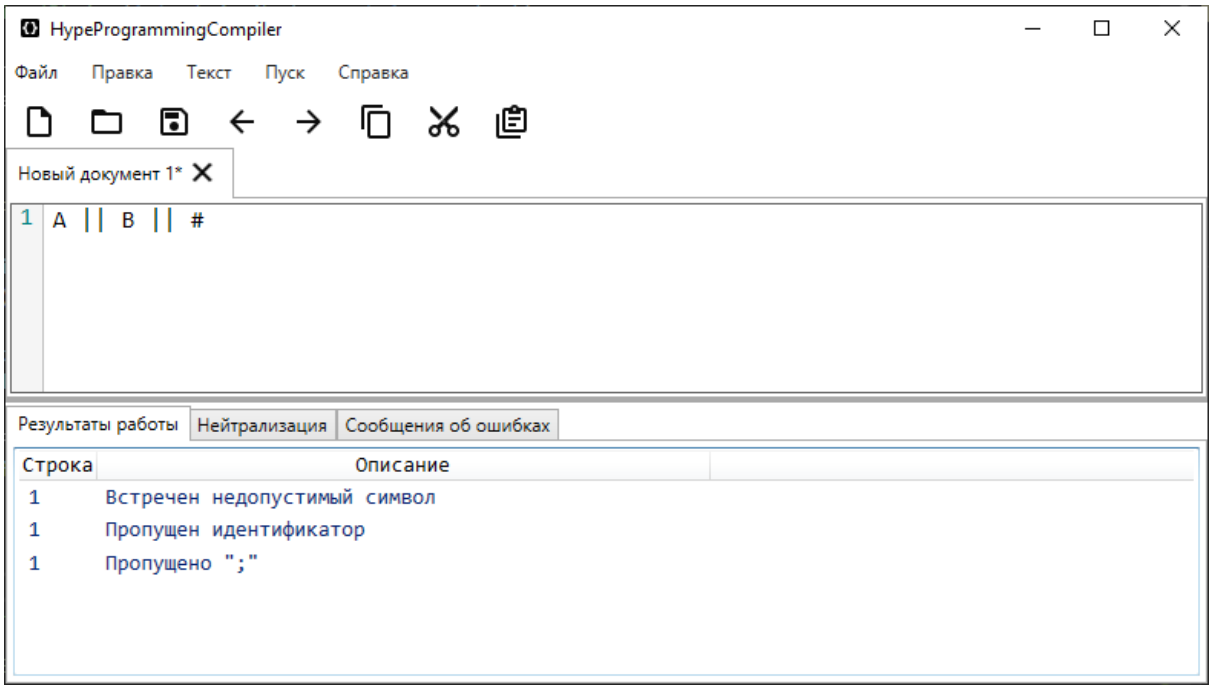


Рисунок 5 – пропуск идентификатора и точки с запятой, ввод недопустимого символа '#'

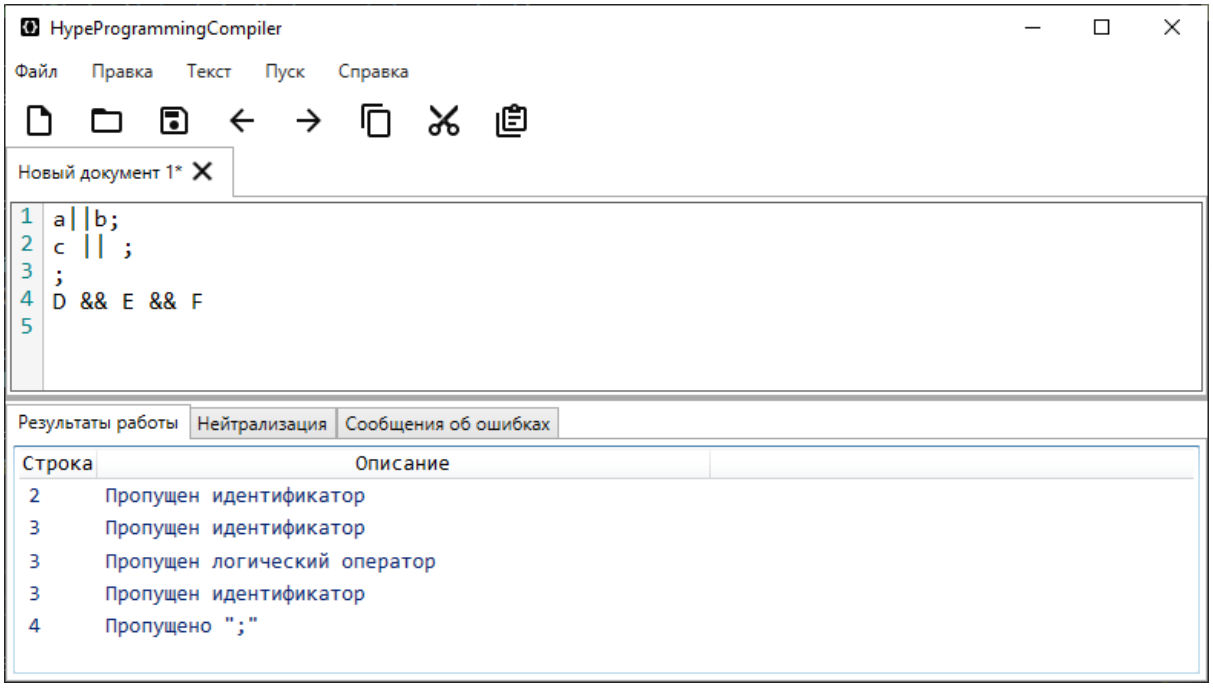


Рисунок 6 – ввод нескольких строк с логическими выражениями

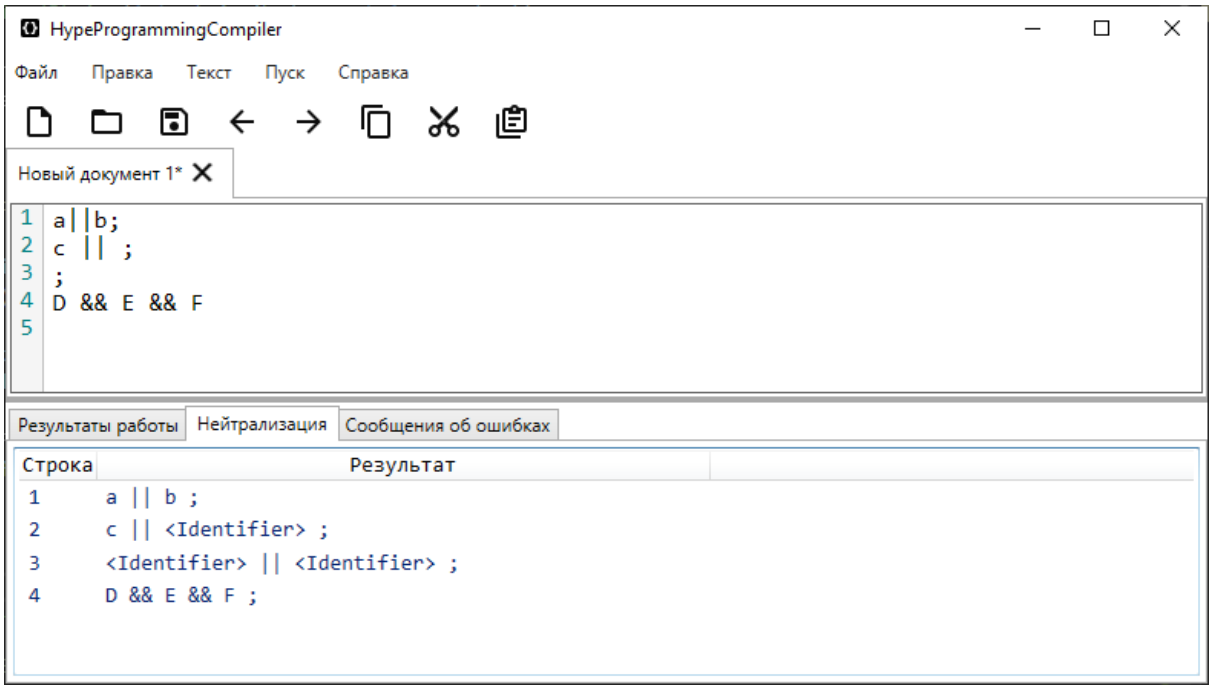


Рисунок 7 – Результат нейтрализации с исправлением ошибок для тех же выражений (см. Рисунок 6)

# 7 ЛИСТИНГ

Lexem.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HypeProgrammingCompiler
{
    public enum LexemType // Перечень типов лексем
    {
        ErrorOperator = -1,
        ErrorToken = 0,
        Identifier = 1,
        Disjunction = 2,
        Conjunction = 3,
        Semicolon = 4
    }

    // Предоставляет хранение свойств токена и доступ к ним
    public class Lexem
    {
        public LexemType Type { get; set; }
        public string Symbol { get; set; }
        public int StringNumber { get; set; }
        public int StartPosition { get; set; }
        public int EndPosition { get; set; }

        public Lexem(LexemType type, string symbol, int stringNumber, int
startPosition, int endPosition)
        {
            Type = type;
            Symbol = symbol;
            StringNumber = stringNumber;
            StartPosition = startPosition;
            EndPosition = endPosition;
        }

        public Lexem(LexemType type, string symbol, int stringNumber)
        {
            Type = type;
            Symbol = symbol;
            StringNumber = stringNumber;
        }
    }
}
```

LexemList.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HypeProgrammingCompiler
{
    // Хранит лист токенов
    class LexemList
    {
        public List<Lexem> lexems = new List<Lexem>();
        private int i = 0;

        public LexemList() { }
        public LexemList(List<Lexem> lexems) { this.lexems = lexems; }

        public Lexem Current // Возврат лексемы на которую указывает индекс
        {
            get
            {
                if (i >= lexems.Count)
                {
                    return new Lexem(0, null, lexems[i - 1].StringNumber, 0,
0);
                }
                return lexems[i];
            }
            set { }
        }

        public bool Next() // Перемещение индекса вперёд
        {
            i++;
            if (i < lexems.Count)
            {
                return true;
            }
            else
                return false;
        }

        public Lexem Prev // Возврат предыдущей лексемы от текщуего индекса
        {
            get
            {
                if (i > 0)
                    return lexems[i - 1];
                else
                    return new Lexem(0, null, 0, 0, 0);
            }
        }

        public void Add(Lexem lexem) // Вставка в конец
```

```
        {
            lexems.Add(lexem);
        }

        public int Count // Возврат числа лексем в списке
        {
            get
            {
                return lexems.Count;
            }
        }

        public void RemoveNext(Lexem lexem) // Удаление лексемы следующей за
текущей
        {
            lexems.RemoveAt(lexems.IndexOf(lexem) + 1);
        }

        public void Insert(Lexem lexem) // Вставка лексемы по текущему индексу
        {
            lexems.Insert(i , lexem);
        }

        public void Replace(Lexem lexem)
        {
            int index = lexems.FindIndex(l => l == Current);

            if (index != -1)
                lexems[index] = lexem;

        }
    }
}
```

Lexer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Text.RegularExpressions;

namespace HypeProgrammingCompiler
{
    class Lexer
    {
        public LexemList lexemList = new LexemList(); // Список лексем для
заполнения
        private List<string> textStrings = new List<string>();
        private int i = 0; // Индекс символа
        private int stringNumber = 0; // Номер строки

        public Lexer(string text)
        {
            textStrings = text.Split("\n").ToList();
            textStrings[textStrings.Count - 1] += "\r";
        }

        public string Print()
        {
            string result = "";

            do
            {
                result += lexemList.Current.Type + " - ";
                result += lexemList.Current.Symbol + " - ";
                result += "S: " + (lexemList.Current.StringNumber) + "\n";
            }
            while (lexemList.Next());

            return result;
        }

        public void Analyze()
        {
            foreach (string textString in textStrings)
            {
                AnalyzeString(textString);
                stringNumber++;
            }
        }

        private void AnalyzeString(string textString)
        {
            i = 0;
            for (; i < textString.Length; i++)
            {
                // Пропуск пробелов перед лексемой
                while (textString[i] == ' ') i++;
```

```
                // Если лексема начинается с буквы, то парсим идентификатор
                if (char.IsLetter(textString[i]))
                {
                    MatchIdentifier(textString);
                    if (textString.Length <= i)
                        break;
                }
                // Иначе парсим дизъюнкцию
                else if (textString[i] == '|')
                {
                    MatchDisjunction(textString);
                    if (textString.Length <= i)
                        break;
                }
                // Иначе парсим конъюнкцию
                else if (textString[i] == '&')
                {
                    MatchConjunction(textString);
                    if (textString.Length <= i)
                        break;
                }
                // Иначе парсим точку с запятой
                else if (textString[i] == ';')
                {
                    MatchSemicolon();
                    if (textString.Length <= i)
                        break;
                }
                // Иначе парсим неизвестный символ
                else
                {
                    if (textString[i] != '\r')
                        MatchErrorToken(textString);
                }
            }

        }

        private void MatchIdentifier(string textString)
        {
            string identifier = "";
            int startPosition = i + 1;

            while (textString[i] != ' ' && textString[i] != ';' &&
textString[i] != '\n' && textString[i] != '\r')
            {
                identifier += textString[i];
                i++;

                if (textString.Length <= i)
                    break;

                if (textString[i] == '|' || textString[i] == '&')
                {
                    if (textString.Length + 1 > i)
```

```
                    {
                        if (textString[i + 1] == '|' || textString[i + 1] ==
'&')
                            break;
                    }
                }

            }
            int endPosition = i;

            string pattern = @"\W";
            if (Regex.IsMatch(identifier, pattern, RegexOptions.ECMAScript))
            {
                lexemList.Add(new Lexem(LexemType.ErrorToken, identifier,
stringNumber + 1, startPosition, endPosition));
            }
            else
                lexemList.Add(new Lexem(LexemType.Identifier, identifier,
stringNumber + 1, startPosition, endPosition));
            i--;
        }

        private void MatchDisjunction(string textString)
        {
            if (textString.Length != i + 1)
            {
                if (textString[i + 1] == '|')
                {
                    lexemList.Add(new Lexem(LexemType.Disjunction, "||",
stringNumber + 1, i + 1, i + 2));
                    i++;
                    return;
                }
            }
            // Парсим некорректный оператор
            string errorToken = "";
            int startPosition = i + 1;

            while (textString[i] != ' ' && textString[i] != ';' &&
textString[i] != '\n' && textString[i] != '\r')
            {
                errorToken += textString[i];
                i++;

                if (textString.Length <= i)
                    break;

                if (textString[i] == '|' || textString[i] == '&')
                {
                    if (textString.Length + 1 > i)
                    {
                        if (textString[i + 1] == '|' || textString[i + 1] ==
'&')
                            break;
                    }
                }
```

```
            }
            int endPosition = i;
            i--;

            lexemList.Add(new Lexem(LexemType.ErrorOperator, errorToken,
stringNumber + 1, i + 1, i + 1));
        }

        private void MatchConjunction(string textString)
        {
            if (textString.Length != i + 1)
            {
                if (textString[i + 1] == '&')
                {
                    lexemList.Add(new Lexem(LexemType.Disjunction, "&&",
stringNumber + 1, i + 1, i + 2));
                    i++;
                    return;
                }
            }
            // Парсим некорректный оператор
            string errorToken = "";
            int startPosition = i + 1;

            while (textString[i] != ' ' && textString[i] != ';' &&
textString[i] != '\n' && textString[i] != '\r')
            {
                errorToken += textString[i];
                i++;

                if (textString.Length <= i)
                    break;

                if (textString[i] == '|' || textString[i] == '&')
                {
                    if (textString.Length + 1 > i)
                    {
                        if (textString[i + 1] == '|' || textString[i + 1] ==
'&')
                            break;
                    }
                }

            }
            int endPosition = i;
            i--;

            lexemList.Add(new Lexem(LexemType.ErrorOperator, errorToken,
stringNumber + 1, i + 1, i + 1));
        }

        private void MatchSemicolon()
        {
            lexemList.Add(new Lexem(LexemType.Semicolon, ";", stringNumber +
1, i + 1, i + 1));
```

```
        }

        private void MatchErrorToken(string textString)
        {

            string errorToken = "";
            int startPosition = i + 1;

            while (textString[i] != ' ' && textString[i] != ';' &&
textString[i] != '\n' && textString[i] != '\r')
            {
                errorToken += textString[i];
                i++;

                if (textString.Length <= i)
                    break;

                if (textString[i] == '|' || textString[i] == '&')
                {
                    if (textString.Length + 1 > i)
                    {
                        if (textString[i + 1] == '|' || textString[i + 1] ==
'&')
                            break;
                    }
                }

            }
            int endPosition = i;
            i--;

            lexemList.Add(new Lexem(LexemType.ErrorToken, errorToken, string-
Number + 1, i + 1, i + 1));
        }
    }
}
```

Parser.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Collections.ObjectModel;

namespace HypeProgrammingCompiler
{
    class Parser
    {
        // Исходный текст
        private string text;
        // Список ошибок для вывода и нейтралищации
        public ObservableCollection<Error> errorList = new ObservableCollec-
tion<Error>();
        // Список лексем - результат декомпозиции текста
        public LexemList lexemList = new LexemList();
        // Исправленные строки
        public List<FixedString> FixedStrings = new List<FixedString>();
        public struct FixedString
        {
            public int StringNumber { get; set; }
            public string Content { get; set; }

            public FixedString(int stringNumber, string content)
            {
                StringNumber = stringNumber;
                Content = content;
            }
        }
        public void PrepareFixedStrings()
        {
            string fixedString = "";
            foreach (var lexem in lexemList.lexems)
            {
                fixedString += lexem.Symbol + " ";
                if (lexem.Symbol == ";")
                {
                    FixedStrings.Add(new FixedString(lexem.StringNumber,
fixedString));
                    fixedString = "";
                }
            }
        }
        // Перечесление кодов ошибок
        public enum ErrorCode
        {
            NoIdentifier = 1,
            NoOperator = 2,
            NoSemicolon = 3,
            InvalidOperator = 4,
            InvalidCharacter = 5
        }
```

```
        //Класс Error - содержит информацию об ошибке и её код
        public class Error
        {
            public string Info { get; set; }
            public ErrorCode Code { get; set; }
            public string Symbol { get; set; }
            public int StringNumber { get; set; }
            public int StartPosition { get; set; }
            public int EndPosition { get; set; }

            public Error(string info, ErrorCode code, string symbol, int
stringNumber, int startPosition, int endPoition)
            {
                Info = info;
                Code = code;
                Symbol = symbol;
                StringNumber = stringNumber;
                StartPosition = startPosition;
                EndPosition = endPoition;
            }
            public Error(string info, ErrorCode code, int stringNumber, int
startPosition, int endPoition)
            {
                Info = info;
                Code = code;
                StringNumber = stringNumber;
                StartPosition = startPosition;
                EndPosition = endPoition;
            }
        }
        // Специальное сообщение об отсутствии ошибок
        public struct NoErrorMessage
        {
            public string Info { get { return "Ошибок не найдено"; } }
        }
        public Parser(string text)
        {
            this.text = text;
            errorList.CollectionChanged += ErrorList_CollectionChanged;
        }
        // Нейтрализация ошибок при их обнаружении
        private void ErrorList_CollectionChanged(object sender, System.Collec-
tions.Specialized.NotifyCollectionChangedEventArgs e)
        {
            Error error = e.NewItems[0] as Error;

            switch (error.Code)
            {
                case ErrorCode.NoIdentifier:
                    lexemList.Insert(new Lexem(LexemType.Identifier, "<Identi-
fier>", lexemList.Current.StringNumber, lexemList.Current.StartPosition, 0));
                    break;
                case ErrorCode.NoOperator:
                    lexemList.Insert(new Lexem(LexemType.Conjunction, "<Opera-
tor>", lexemList.Current.StringNumber, lexemList.Current.StartPosition, 0));
```

```
                    break;
                case ErrorCode.NoSemicolon:
                    lexemList.Insert(new Lexem(LexemType.Semicolon, ";", lex-
emList.Current.StringNumber, lexemList.Current.StartPosition, 0));
                    break;
                case ErrorCode.InvalidOperator:
                    if (lexemList.Current.Symbol[0] == '|')
                        lexemList.Insert(new Lexem(LexemType.Conjunction,
"||", lexemList.Current.StringNumber, lexemList.Current.StartPosition, 0));
                    else
                        lexemList.Insert(new Lexem(LexemType.Conjunction,
"&&", lexemList.Current.StringNumber, lexemList.Current.StartPosition, 0));

                    lexemList.RemoveNext(lexemList.Current);
                    break;
            }
        }
        private bool IsErrorLexem(Lexem lexem) // Предикат определения
недопустимых лексем и их удаления
        {
            if (lexem.Type == LexemType.ErrorToken)
            {
                errorList.Add(new Error("Встречен недопустимый символ: \"" +
lexem.Symbol + "\"", ErrorCode.InvalidCharacter, lexem.Symbol, lexem.String-
Number, lexem.StartPosition, lexem.EndPosition));
                return true;
            }
            return false;
        }
        public void Parse()
        {
            Lexer lexer = new Lexer(text);
            lexer.Analyze(); // Декомпозиция текста на лексемы
            lexemList = lexer.lexemList;

            // Нейтрализация недопустимых символов
            lexemList.lexems.RemoveAll(IsErrorLexem);

            if (lexemList.Count > 0)
            {
                do
                {
                    // Если пропущен идентификатор
                    if (lexemList.Current.Type != LexemType.Identifier)
                        errorList.Add(new Error("Пропущен идентификатор", Er-
rorCode.NoIdentifier, lexemList.Current.StringNumber, lexemList.Cur-
rent.StartPosition, lexemList.Current.EndPosition));

                    lexemList.Next();
                    // Если оператор пропущен или некорректен
                    if (lexemList.Current.Type != LexemType.Disjunction &&
                        lexemList.Current.Type != LexemType.Conjunction)
                    {
                        if (lexemList.Current.Type == LexemType.ErrorOperator)
                            errorList.Add(new Error("Некорректный логический
оператор: \"" + lexemList.Current.Symbol + "\"", ErrorCode.InvalidOperator,
```

```
lexemList.Current.Symbol, lexemList.Current.StringNumber, lexemList.Cur-
rent.StartPosition, lexemList.Current.EndPosition));
                        else
                            errorList.Add(new Error("Пропущен логический
оператор", ErrorCode.NoOperator, lexemList.Prev.StringNumber, lexemList.Cur-
rent.StartPosition, lexemList.Current.EndPosition));
                    }

                    lexemList.Next();
                    //Если пропущен идентификатор
                    if (lexemList.Current.Type != LexemType.Identifier)
                        errorList.Add(new Error("Пропущен идентификатор", Er-
rorCode.NoIdentifier, lexemList.Current.StringNumber, lexemList.Cur-
rent.StartPosition, lexemList.Current.EndPosition));

                    // Итерация * (Замыкание Клини)
                    while (lexemList.Next() && lexemList.Current.Type != Lex-
emType.Semicolon)
                    {
                        if (lexemList.Current.Type != LexemType.Disjunction &&
                            lexemList.Current.Type != LexemType.Conjunction)
                        {
                            if (lexemList.Current.Type == LexemType.ErrorOper-
ator)
                                errorList.Add(new Error("Некорректный
логический оператор: \"" + lexemList.Current.Symbol + "\"", ErrorCode.In-
validOperator, lexemList.Current.Symbol, lexemList.Current.StringNumber, lex-
emList.Current.StartPosition, lexemList.Current.EndPosition));
                            else
                                errorList.Add(new Error("Пропущен логический
оператор", ErrorCode.NoOperator, lexemList.Prev.StringNumber,
lexemList.Current.StartPosition, lexemList.Current.EndPosition));
                        }

                        lexemList.Next();
                        if (lexemList.Current.Type != LexemType.Identifier)
                            errorList.Add(new Error("Пропущен идентификатор",
ErrorCode.NoIdentifier, lexemList.Current.StringNumber, lexemList.Cur-
rent.StartPosition, lexemList.Current.EndPosition));
                    }

                    // Если пропущено ";"
                    if (lexemList.Current.Type != LexemType.Semicolon)
                        errorList.Add(new Error("Пропущено \";\"", Error-
Code.NoSemicolon, lexemList.Current.StringNumber, lexemList.Current.StartPosi-
tion, lexemList.Current.EndPosition));
                }
                while (lexemList.Next()); // Цикл до последней лексемы
включительно
            }

            PrepareFixedStrings();
        }
    }
}
```

MainWindow.xaml.cs

```
using System;
using Microsoft.Win32;
using System.Security;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using FastColoredTextBoxNS;
using System.Windows.Forms.Integration;
using System.IO;
using System.Diagnostics;
using System.Net;

namespace HypeProgrammingCompiler
{
    public partial class MainWindow : Window
    {
        List<bool> isSaved = new List<bool>(); //Флаги сохранений
        List<bool> isExist = new List<bool>(); //Флаги существований в
файловой системе
        List<string> filePath = new List<string>(); //Полные имена в файловой
системе

        public MainWindow()
        {
            InitializeComponent();
            AddKeys();
            AddPage(null, null);
        }

        // Горячие калвиши
        private void AddKeys()
        {
            RoutedCommand openCommand = new RoutedCommand();
            openCommand.InputGestures.Add(new KeyGesture(Key.O, Modi-
fierKeys.Control));
            CommandBindings.Add(new CommandBinding(openCommand, Open));

            RoutedCommand addCommand = new RoutedCommand();
            addCommand.InputGestures.Add(new KeyGesture(Key.N, Modi-
fierKeys.Control));
            CommandBindings.Add(new CommandBinding(addCommand, AddPage));

            RoutedCommand saveCommand = new RoutedCommand();
            saveCommand.InputGestures.Add(new KeyGesture(Key.S, Modi-
fierKeys.Control));
            CommandBindings.Add(new CommandBinding(saveCommand, Save));
```

```
        }

        private void AddPage(object sender, RoutedEventArgs e)
        {
            //Добавление иконки для конпки закрытия вкладки
            Image image = new Image();
            image.Source = new BitmapImage(new Uri(@"/Resources/close.png",
UriKind.RelativeOrAbsolute));

            //Кнопка закрытия вкладки
            Button closeDocumentButton = new Button { Content = image, Border-
Thickness = new Thickness(0), Background = Brushes.Transparent};
            closeDocumentButton.Click += CloseDocumentButton_Click;

            //Контейнер хранения заголовка вкладки и кнопки закрытия
            StackPanel stackPanel = new StackPanel();
            stackPanel.Orientation = Orientation.Horizontal;
            stackPanel.Children.Add(new TextBlock { Text = "Новый документ " +
(InputTabControl.Items.Count + 1).ToString(), VerticalAlignment = VerticalA-
lignment.Center});
            stackPanel.Children.Add(closeDocumentButton);

            //Область ввода текста
            FastColoredTextBox fastColoredTextBox = new FastColoredTextBox();
            fastColoredTextBox.BorderStyle = System.Windows.Forms.Border-
Style.FixedSingle;
            fastColoredTextBox.TextChanged += FastColoredTextBox_TextChanged;
            fastColoredTextBox.Font = new System.Drawing.Font("Consolas", 12);
            fastColoredTextBox.Zoom = 100;
            WindowsFormsHost windowsFormsHost = new WindowsFormsHost();
            windowsFormsHost.Child = fastColoredTextBox;

            //Добавление новой вкладки
            TabItem tabItem = new TabItem()
            {
                Header = stackPanel, //Заголовок
                Content = windowsFormsHost, //Текстовое поле
                IsSelected = true
            };
            InputTabControl.Items.Add(tabItem);

            //Добавление список для отслеживания изменений
            isSaved.Add(true);
            isExist.Add(false);
            filePath.Add("");
        }

        private void CloseDocumentButton_Click(object sender, RoutedEventArgs
e)
        {
            //Получение вкладки которую необходимо закрыть
            Button button = sender as Button;
            StackPanel stackPanel = button.Parent as StackPanel;
            TabItem tabItem = stackPanel.Parent as TabItem;

            //Получаем индекс вкладки которую необходимо закрыть
```

```
            int tabIndexToClose = InputTabControl.Items.IndexOf(tabItem);

            //Если изменения сохранены
            if (isSaved[tabIndexToClose])
            {
                InputTabControl.Items.Remove(tabItem); //Закрыть
                isSaved.RemoveAt(tabIndexToClose); //Перестать отслеживать
изменения
                isExist.RemoveAt(tabIndexToClose);
                filePath.RemoveAt(tabIndexToClose);
            }
            else
            {
                MessageBoxResult messageBoxResult = MessageBox.Show("В
документе были сделаны изменения. Сохранить их?", "Внимание", MessageBoxBut-
ton.YesNoCancel, MessageBoxImage.Question);
                switch (messageBoxResult)
                {
                    case MessageBoxResult.No:
                        InputTabControl.Items.Remove(tabItem); //Закрыть
                        isSaved.RemoveAt(tabIndexToClose); //Перестать
отслеживать изменения
                        isExist.RemoveAt(tabIndexToClose);
                        filePath.RemoveAt(tabIndexToClose);
                        break;

                    case MessageBoxResult.Yes:
                        Save(sender, e);
                        break;
                }
            }
        }

        // Звёздочки
        private void FastColoredTextBox_TextChanged(object sender, FastCol-
oredTextBoxNS.TextChangedEventArgs e)
        {
            if (isSaved[InputTabControl.SelectedIndex])
            {
                isSaved[InputTabControl.SelectedIndex] = false;
                TabItem tabItem = InputTabControl.SelectedItem as TabItem;
                StackPanel stackPanel = tabItem.Header as StackPanel;
                (stackPanel.Children[0] as TextBlock).Text += "*"; ;
            }
        }

        private void NewFileButton_Click(object sender, RoutedEventArgs e)
        {
            AddPage(null, null);
        }

        private void Save(object sender, RoutedEventArgs e)
        {
            //Если файл существует, сохраняем
            if (isExist[InputTabControl.SelectedIndex])
            {
```

```
                try
                {
                    TabItem tabItem = InputTabControl.SelectedItem as TabItem;
                    WindowsFormsHost windowsFormsHost = tabItem.Content as
WindowsFormsHost;
                    FastColoredTextBox fastColoredTextBox = win-
dowsFormsHost.Child as FastColoredTextBox;

                    File.WriteAllText(filePath[InputTabControl.SelectedIndex],
fastColoredTextBox.Text);

                    StackPanel stackPanel = tabItem.Header as StackPanel;
                    (stackPanel.Children[0] as TextBlock).Text = (stack-
Panel.Children[0] as TextBlock).Text.Trim('*');
                }
                catch (ArgumentException exception) { ErrorPrint("Данный путь
недопустим или содержит недопустимые символы"); }
                catch (PathTooLongException exception) { ErrorPrint("Путь или
имя файла превышают допустимую длину"); }
                catch (DirectoryNotFoundException exception) { Error-
Print("Указан недопустимый путь (например, он ведет на несопоставленный
диск)"); }
                catch (IOException exception) { ErrorPrint("При открытии файла
произошла ошибка ввода-вывода"); }
                catch (UnauthorizedAccessException exception) { Error-
Print(""); }
                catch (NotSupportedException exception) { ErrorPrint("Неверный
формат файла"); }
                catch (SecurityException exception) { ErrorPrint("Неверный
формат файла"); }
            }
            else //Иначе - сохраняем как...
            {
                SaveAs(sender, e);
            }
            isSaved[InputTabControl.SelectedIndex] = true;
        }

        private void SaveAs(object sender, RoutedEventArgs e)
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.AddExtension = true;
            saveFileDialog.Filter = "hpl files (*.hpl)|*.hpl|txt files
(*.txt)|*.txt|cs files (*.cs)|*.cs|cpp files (*.cpp)|*.cpp|h files
(*.h)|*.h|py files (*.py)|*.py|html files (*.html)|*.html|js files
(*.js)|*.js|php files (*.php)|*.php";
            saveFileDialog.RestoreDirectory = true;
            if (saveFileDialog.ShowDialog() == true)
            {
                TabItem tabItem = InputTabControl.SelectedItem as TabItem;
                WindowsFormsHost windowsFormsHost = tabItem.Content as Win-
dowsFormsHost;
                FastColoredTextBox fastColoredTextBox = windowsFormsHost.Child
as FastColoredTextBox;
```

```
                File.WriteAllText(saveFileDialog.FileName, fastColoredText-
Box.Text);
                StackPanel stackPanel = tabItem.Header as StackPanel;

                (stackPanel.Children[0] as TextBlock).Text = saveFileDia-
log.SafeFileName;

                isExist[InputTabControl.SelectedIndex] = true;
                filePath[InputTabControl.SelectedIndex] = saveFileDialog.File-
Name;
            }
        }

        private void OpenFile(string fileName, string safeFileName)
        {
            foreach (string f in filePath)
            {
                if (f == fileName)
                {
                    throw new IOException();
                }
            }

            AddPage(null, null);
            TabItem tabItem = InputTabControl.SelectedItem as TabItem;
            WindowsFormsHost windowsFormsHost = tabItem.Content as Win-
dowsFormsHost;
            FastColoredTextBox fastColoredTextBox = windowsFormsHost.Child as
FastColoredTextBox;
            fastColoredTextBox.Text = File.ReadAllText(fileName);

            StackPanel stackPanel = tabItem.Header as StackPanel;
            (stackPanel.Children[0] as TextBlock).Text = safeFileName;

            isSaved[InputTabControl.SelectedIndex] = true;
            isExist[InputTabControl.SelectedIndex] = true;
            filePath[InputTabControl.SelectedIndex] = fileName;
        }

        private void Open(object sender, RoutedEventArgs e)
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Filter = "hpl files (*.hpl)|*.hpl|txt files
(*.txt)|*.txt|cs files (*.cs)|*.cs|cpp files (*.cpp)|*.cpp|h files
(*.h)|*.h|py files (*.py)|*.py|html files (*.html)|*.html|js files
(*.js)|*.js|php files (*.php)|*.php";
            if (openFileDialog.ShowDialog() == true)
            {
                try
                {
                    OpenFile(openFileDialog.FileName, openFileDialog.SafeFile-
Name);
                }
                catch (FileFormatException exception) { ErrorPrint("Неверный
формат файла"); }
```

```
                catch (FileLoadException exception) { ErrorPrint("Файл не
может быть загружен"); }
                catch (FileNotFoundException exception) { ErrorPrint("Файл не
найден"); }
                catch (IOException exception) { ErrorPrint(String.Format("Файл
{0} уже загружен", openFileDialog.SafeFileName)); }
            }
        }

        private void ErrorPrint(string message)
        {
            ErrorTextBlock.Text = message;
            (OutputTabControl.Items[1] as TabItem).IsSelected = true;
        }

        private void RemoveTab(object sender, RoutedEventArgs e)
        {
            TabItem tabItem = InputTabControl.SelectedItem as TabItem;
            WindowsFormsHost windowsFormsHost = tabItem.Content as Win-
dowsFormsHost;
            FastColoredTextBox fastColoredTextBox = windowsFormsHost.Child as
FastColoredTextBox;

            //Получаем индекс вкладки которую необходимо закрыть
            int tabIndexToClose = InputTabControl.Items.IndexOf(tabItem);

            //Если изменения сохранены
            if (isSaved[tabIndexToClose])
            {
                InputTabControl.Items.Remove(tabItem); //Закрыть
                isSaved.RemoveAt(tabIndexToClose); //Перестать отслеживать
изменения
                isExist.RemoveAt(tabIndexToClose);
                filePath.RemoveAt(tabIndexToClose);
            }
            else
            {
                MessageBoxResult messageBoxResult = MessageBox.Show("В
документе были сделаны изменения. Сохранить их?", "Внимание", MessageBoxBut-
ton.YesNoCancel, MessageBoxImage.Question);
                switch (messageBoxResult)
                {
                    case MessageBoxResult.No:
                        InputTabControl.Items.Remove(tabItem); //Закрыть
                        isSaved.RemoveAt(tabIndexToClose); //Перестать
отслеживать изменения
                        isExist.RemoveAt(tabIndexToClose);
                        filePath.RemoveAt(tabIndexToClose);
                        break;

                    case MessageBoxResult.Yes:
                        Save(sender, e);
                        break;
                }
            }
        }
```

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

        private void Undo(object sender, RoutedEventArgs e)
        {
            TabItem tabItem = InputTabControl.SelectedItem as TabItem;
            WindowsFormsHost windowsFormsHost = tabItem.Content as Win-
dowsFormsHost;
            FastColoredTextBox fastColoredTextBox = windowsFormsHost.Child as
FastColoredTextBox;
            fastColoredTextBox.Undo();
        }

        private void Redo(object sender, RoutedEventArgs e)
        {
            TabItem tabItem = InputTabControl.SelectedItem as TabItem;
            WindowsFormsHost windowsFormsHost = tabItem.Content as Win-
dowsFormsHost;
            FastColoredTextBox fastColoredTextBox = windowsFormsHost.Child as
FastColoredTextBox;
            fastColoredTextBox.Redo();
        }

        private void Cut(object sender, RoutedEventArgs e)
        {
            TabItem tabItem = InputTabControl.SelectedItem as TabItem;
            WindowsFormsHost windowsFormsHost = tabItem.Content as Win-
dowsFormsHost;
            FastColoredTextBox fastColoredTextBox = windowsFormsHost.Child as
FastColoredTextBox;
            fastColoredTextBox.Cut();
        }

        private void Copy(object sender, RoutedEventArgs e)
        {
            TabItem tabItem = InputTabControl.SelectedItem as TabItem;
            WindowsFormsHost windowsFormsHost = tabItem.Content as Win-
dowsFormsHost;
            FastColoredTextBox fastColoredTextBox = windowsFormsHost.Child as
FastColoredTextBox;
            fastColoredTextBox.Copy();
        }

        private void Insert(object sender, RoutedEventArgs e)
        {
            TabItem tabItem = InputTabControl.SelectedItem as TabItem;
            WindowsFormsHost windowsFormsHost = tabItem.Content as Win-
dowsFormsHost;
            FastColoredTextBox fastColoredTextBox = windowsFormsHost.Child as
FastColoredTextBox;
            fastColoredTextBox.InsertText(Clipboard.GetText());
        }
```

```
        private void Delete(object sender, RoutedEventArgs e)
        {
            TabItem tabItem = InputTabControl.SelectedItem as TabItem;
            WindowsFormsHost windowsFormsHost = tabItem.Content as Win-
dowsFormsHost;
            FastColoredTextBox fastColoredTextBox = windowsFormsHost.Child as
FastColoredTextBox;
            fastColoredTextBox.ClearSelected();
        }

        private void SelectAll(object sender, RoutedEventArgs e)
        {
            TabItem tabItem = InputTabControl.SelectedItem as TabItem;
            WindowsFormsHost windowsFormsHost = tabItem.Content as Win-
dowsFormsHost;
            FastColoredTextBox fastColoredTextBox = windowsFormsHost.Child as
FastColoredTextBox;
            fastColoredTextBox.SelectAll();
        }

        private void ShowInfo(object sender, RoutedEventArgs e)
        {
            MessageBox.Show("HypeProgrammingCompiler\n\n(c) 2022 HPS Андрей
Мазуров АВТ-912 АВТФ\ngithub.com/gore-stepanyan/HypeProgrammingCompiler", "О
программе", MessageBoxButton.OK, MessageBoxImage.Information);

        }

        private void OnClosing(object sender, System.ComponentModel.CancelEv-
entArgs e)
        {
            foreach (TabItem tabItem in InputTabControl.Items)
            {
                if (!isSaved[InputTabControl.SelectedIndex])
                {
                    tabItem.IsSelected = true;
                    MessageBoxResult messageBoxResult = MessageBox.Show("В
документе были сделаны изменения. Сохранить их?", "Внимание", MessageBoxBut-
ton.YesNoCancel, MessageBoxImage.Information);
                    switch (messageBoxResult)
                    {
                        case MessageBoxResult.Yes: { Save(null, null); break;
}
                        case MessageBoxResult.Cancel: { e.Cancel = true; re-
turn; }
                    }
                }
            }
        }

        private void ShowManual(object sender, RoutedEventArgs e)
        {
            Process.Start(new ProcessStartInfo("cmd", $"/c start https://gore-
stepanyan.github.io/info/info.pdf"));
        }
```

```
        private void Run(object sender, RoutedEventArgs e)
        {
            TabItem tabItem = InputTabControl.SelectedItem as TabItem;
            WindowsFormsHost windowsFormsHost = tabItem.Content as Win-
dowsFormsHost;
            FastColoredTextBox fastColoredTextBox = windowsFormsHost.Child as
FastColoredTextBox;

            Lexer lexer = new Lexer(fastColoredTextBox.Text);
            lexer.Analyze();
            LexemListTextBlock.Text = lexer.Print();

            Parser parser = new Parser(fastColoredTextBox.Text);
            parser.Parse();

            OutputListView.Items.Clear();
            if (parser.errorList.Count == 0)
                OutputListView.Items.Add(new Parser.NoErrorMessage());
            else
                foreach (Parser.Error error in parser.errorList)
                {

                    OutputListView.Items.Add(error);
                }

            FixedOutputListView.Items.Clear();
            foreach (var fixedString in parser.FixedStrings)
            {
                FixedOutputListView.Items.Add(fixedString);
            }
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            TabItem tabItem = InputTabControl.SelectedItem as TabItem;
            WindowsFormsHost windowsFormsHost = tabItem.Content as Win-
dowsFormsHost;
            FastColoredTextBox fastColoredTextBox = windowsFormsHost.Child as
FastColoredTextBox;

            fastColoredTextBox.Focus();
        }

        private void Task(object sender, RoutedEventArgs e)
        {
            Process.Start(new ProcessStartInfo("cmd", $"/c start https://gore-
stepanyan.github.io/CourseWork/Work.pdf#page=3"));
        }

        private void Grammar(object sender, RoutedEventArgs e)
        {
            Process.Start(new ProcessStartInfo("cmd", $"/c start https://gore-
stepanyan.github.io/CourseWork/Work.pdf#page=4"));
        }

        private void Classification(object sender, RoutedEventArgs e)
```

```
        {
            Process.Start(new ProcessStartInfo("cmd", $"/c start https://gore-
stepanyan.github.io/CourseWork/Work.pdf#page=5"));
        }

        private void AnalyzeMethod(object sender, RoutedEventArgs e)
        {
            Process.Start(new ProcessStartInfo("cmd", $"/c start https://gore-
stepanyan.github.io/CourseWork/Work.pdf#page=6"));
        }

        private void Neutralization(object sender, RoutedEventArgs e)
        {
            Process.Start(new ProcessStartInfo("cmd", $"/c start https://gore-
stepanyan.github.io/CourseWork/Work.pdf#page=7"));
        }

        private void Test(object sender, RoutedEventArgs e)
        {
            Process.Start(new ProcessStartInfo("cmd", $"/c start https://gore-
stepanyan.github.io/CourseWork/Work.pdf#page=8"));
        }

        private void Literature(object sender, RoutedEventArgs e)
        {
            Process.Start(new ProcessStartInfo("cmd", $"/c start https://gore-
stepanyan.github.io/CourseWork/Work.pdf#page=21"));
        }

        private void Listing(object sender, RoutedEventArgs e)
        {
            Process.Start(new ProcessStartInfo("cmd", $"/c start https://gore-
stepanyan.github.io/CourseWork/Work.pdf#page=11"));
        }
    }
}
```

# 8 ЛИТЕРАТУРА


1. Шорников Ю.В. Теория и практика языковых процессоров: Учеб. пособие. – Новосибирск: Изд-во НГТУ, 2004. – 208 с.
2. Руководство по WPF // METANIT.COM сайт о программировании : [электронный ресурс]. – 2021. – URL: https://metanit.com/sharp/wpf/ (дата обращения: 14.04.2022).
3. Полное руководство по языку программирования C# 10 и платформе .NET 6 // METANIT.COM сайт о программировании : [электронный ресурс]. – 2022. – URL: https://metanit.com/sharp/wpf/ (дата обращения: 9.03.2022).
4. Теория формальных языков и компиляторов [электронный ресурс] . – 2021. – URL: https://dispace.edu.nstu.ru/didesk/course/show/8594 (дата обращения: 04.03.2022)