

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«Забайкальский государственный университет»  
(ФГБОУ ВО «ЗабГУ»)

Факультет энергетический

Кафедра информатики, вычислительной техники и прикладной математики

## КУРСОВАЯ РАБОТА

по Теории языков программирования и методам трансляции

на тему «Программная реализация транслятора Pascal-> Блок-схема-> C»

*Отлично*  
*С. Коган*

Выполнили ст. гр. ИВТ-18  
Лавров Р.В.  
Сарманова В.В.

Проверил доцент кафедры ИВТ и ПМ  
Коган Е.С.

Чита

2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ  
федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Забайкальский государственный университет»  
(ФГБОУ ВО «ЗабГУ»)  
Факультет энергетический  
Кафедра информатики, вычислительной техники и прикладной математики

**ЗАДАНИЕ**  
на курсовую работу


По дисциплине: Теория языков программирования и методы трансляции

Студентам: Сармановой В.В., Лаврову Р.В.

Специальности (направления подготовки): 09.03.01 Информатика и  
вычислительная техника

- 1 Тема курсовой работы: «Программная реализация транслятора Pascal-> Блок-  
схема-> С»
- 2 Срок подачи студентом законченной работы: 23.12.2021
- 3 Исходные данные к работе:
  1. Описание предметной области;
  2. «Общие требования к построению и оформлению учебной текстовой  
документации» (МИ 01-02-2018)».

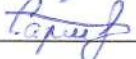
Дата выдачи задания: 15.09.2021

Руководитель курсовой работы  /Коган Е.С./  
(подпись, расшифровка подписи)

Задание принял к исполнению

«15» сентября 2021 г.

Подпись студента  /Лавров Р.В.

Подпись студента  /Сарманова В.В.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ  
федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Забайкальский государственный университет»  
(ФГБОУ ВО «ЗабГУ»)  
Факультет энергетический  
Кафедра информатики, вычислительной техники и прикладной математики

## **ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

По дисциплине Теория языков программирования и методы трансляции

На тему «Программная реализация транслятора Pascal-> Блок-схема-> С»

Выполнили студенты группы ИВТ–18: Сарманова В. В., Лавров Р.В.

Руководитель работы: доцент кафедры ИВТ и ПМ Коган Е.С.

Чита

2021

## РЕФЕРАТ

Пояснительная записка 25с., 3 рис., 0 табл., 7 источников, 1 прил.

КОНВЕРТОР, PASCAL, ЯЗЫК C, АЛГОРИТМ, ПЕРЕМЕННАЯ, ЦИКЛ, УСЛОВИЕ, БЛОК-СХЕМА.

Для достижения цели данной работы необходимо выполнить следующие задачи:

- реализовать модули программного продукта;
- создать интерфейс взаимодействия пользователя и программы;
- реализовать программу конвертации из языка Pascal в Блок-схему и на язык C.

Приложение позволяет конвертировать код, написанный на языке Pascal в Блок-схему, а из неё в код, написанный на языке C. Для создания программы был использован метод языка C#, использующий регулярные выражения, которые являются универсальными для работы с шаблонами.

Интерфейс был создан при помощи языка C#.

## СОДЕРЖАНИЕ

1. Теоретическая часть .....	7
1.1 Структура языка Pascal.....	7
1.2. Структура Блок-схем .....	7
1.3. Структура языка C .....	9
1.4. Перевод из языка Pascal в Блок-схему и в язык C .....	11
1.4.1. Функции обработки .....	9
1.4.2. Анализ ошибок.....	11
2. Практическая часть.....	12
2.1. Программная реализация .....	12
ЗАКЛЮЧЕНИЕ .....	15
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....	16
ПРИЛОЖЕНИЕ .....	17

## ВВЕДЕНИЕ

Целью данной курсовой работы является разработка конвертора с языка Pascal в Блок-схему, а из неё на язык C. Для достижения поставленной цели было решено разработать программное средство, способное реализовать процесс конвертации.

Для создания программы был использован метод языка C#, использующий регулярные выражения, которые являются универсальными для работы с шаблонами.

Интерфейс был создан при помощи языка разметки C#.

# 1. Теоретическая часть

Для того, чтобы реализовать конвертор из языка Pascal в Блок-схему и в язык С нам потребуется детально рассмотреть оба языка программирования и правила построения блок-схем, найти соответствие между ключевыми словами и конструкциями.

## 1.1 Структура языка Pascal

Язык Pascal – это чисто процедурный язык программирования, часто использующийся для обучения структурному программированию. Особенности языка являются строгая типизация и наличие средств структурного (процедурного) программирования. Язык предоставляет ряд встроенных структур данных: записи, массивы, файлы, множества и указатели.

Для того, чтобы описать алгоритм будущей программы используются следующие ключевые слова:

1. Program – это ключевое слово начало программы;
2. Var – это ключевое слово используется для описания переменных, которые будут использованы в программе;
3. Const – это ключевое слово, после которого идет перечисление констант;
4. Begin – обозначение начала алгоритма;
5. End – обозначение конца алгоритма.
6. Array – ключевое слово для обозначения массива.

Для правильного описания переменных в алгоритме предусмотрены ключевые слова для обозначения их типов:

1. Integer – это целочисленный тип данных;
2. Real – эта лексема используется для обозначения вещественного типа данных;

3. String – это ключевое слово требуется для описания символьных переменных;

4. Boolean – это ключевое слово говорит о том, что переменная логическая.

Циклические конструкции в алгоритмическом языке также используются, для их реализации используются такие ключевые слова:

1. Begin и end– используются для обозначения начала и конца цикла;
2. While – эта лексема используется для циклов с предусловием.
3. For – это ключевое слово используется для цикла со счётчиком.
4. Repeat, until - эта лексема используется для циклов с постусловием.
5. To, do, downto – ключевые слова являются параметрическими.

Для реализации условий в алгоритмическом языке используются ключевые слова:

1. If – используется перед условием, после которого будет происходить выполнение алгоритма;
2. Then – это ключевое слово, указывающее на действие, которое будет выполняться после невыполнения условия, указанное после «if»;
3. Else– – это ключевое слово, указывающее на действие, которое будет выполняться после невыполнения условия, указанное после «if»;
4. And, or, not, <,>, =, <=,>=, <> – логические знаки и выражений используются стандартные.

Ввод и вывод обозначается специальными словами Write, Writeln и Read, Readln соответственно.



## 1.2. Структура Блок-схем

Блок-схема — распространённый тип схем (графических моделей), описывающих алгоритмы или процессы, в которых отдельные шаги изображаются в виде блоков различной формы, соединённых между собой линиями, указывающими направление последовательности.

Название блока	Обозначение	Назначение блока
Терминатор		Начало, завершение программы или подпрограммы
Процесс		Обработка данных (вычисления, пересылки и т. п.)
Данные		Операции ввода-вывода
Решение		Ветвления, выбор, итерационные и поисковые циклы

Рисунок 1 – Элементы блок-схемы

## 1.3. Структура языка C

C – компилируемый статически типизированный язык программирования общего назначения, разработанный в 1969—1973 годах сотрудником Bell Labs Деннисом Ритчи как развитие языка Би. Первоначально был разработан для реализации операционной системы UNIX, но впоследствии был перенесён на множество других платформ. Согласно дизайну языка, его конструкции близко сопоставляются типичным машинным инструкциям, благодаря чему он нашёл применение в проектах, для которых был свойственен язык ассемблера, в том числе как в операционных системах, так и в различных прикладном программном обеспечении для множества устройств — от суперкомпьютеров до

встраиваемых систем. Язык программирования С оказал существенное влияние на развитие индустрии программного обеспечения, а его синтаксис стал основой для таких языков программирования, как C++, C#, Java и Objective-C.

Каждый блок заключается в фигурные скобки { }. Основным блоком является функция, имеющая имя main ().

Как и в любом языке программирования в языке С присутствуют ключевые слова, отвечающие за описание типов переменных:

1. Int – требуется для указания на то, что переменная целочисленного типа;
2. Float – это ключевое слово, использующаяся для указания вещественного типа данных;
3. Char[] – эта лексема указывает на символьный тип. Строку;
4. Char – указывает на один символ;
5. Bool– Логическая переменная, хранящая true или false.

Условные операторы в языке С представлены ключевым словом If, далее идёт блок действий в фигурных скобках.

Для описания циклических конструкций языке С используются следующие лексемы:

1. While –эта лексема используется для циклов с предусловием;
2. Do...while – эта конструкция, используемая для циклов с постусловием;
3. For – это ключевое слово требуется для циклов с заданным числом повторений.

Для логических выражений в языке С используются символы:

1. &&– это логическое И;
2. || – это логическое ИЛИ;
3. ! –это логическое НЕ.

Для ввода и вывода на экран существуют printf() для вывода и scanf() для ввода.

## **1.4. Перевод из языка Pascal в Блок-схему и в язык C**

### **1.4.1. Функции обработки**

Лексема – это структурная единица языка, которая состоит из элементарных символов языка и не содержит в своём составе других структурных единиц языка. Лексемами языков программирования являются идентификаторы, константы, ключевые слова языка, знаки операций и т.п.

Функции обработки представляют собой анализ лексем, найденных в тексте исходной программы лексем. Этот перечень лексем можно представить в виде списка лексем.

Каждой лексеме в списке лексем соответствует некая уникальная лексема другого языка, зависящая от типа лексемы.

Результатом работы функций обработки является перечень всех найденных в тексте исходной программы лексем.

### **1.4.2. Анализ ошибок**

Анализ кода - один из самых надежных методов выявления дефектов. Он заключается в чтении программой исходного кода и высказывании рекомендаций по его улучшению. В процессе чтения кода выявляются ошибки или участки кода, которые являются ошибочными.

Функции обработки ошибок получают строку лексем от лексем исходного текста программы и проверяют, соответствует ли эта строка грамматике исходного языка.

В обработку ошибок входит генерация сообщений обо всех выявленных ошибках, причём достаточно внятных и полных, а кроме того, функции обработки ошибок должны уметь обрабатывать обычные, часто встречающиеся ошибки и продолжать работу с оставшейся частью программы.

## 2. Практическая часть

### 2.1. Программная реализация

Конвертор из языка Pascal в Блок-схему и в язык C был частично реализован. В данной программе получилось реализовать конвертер из языка Pascal в язык C, а из языка C в Блок-схему.

Конвертация основана на соответствии ключевых слов и конструкций из данных языков программирования. При считывании слов, введенных языке Pascal, программа ищет им соответствие на языке C, а потом строит Блок-схему по коду на языке C.

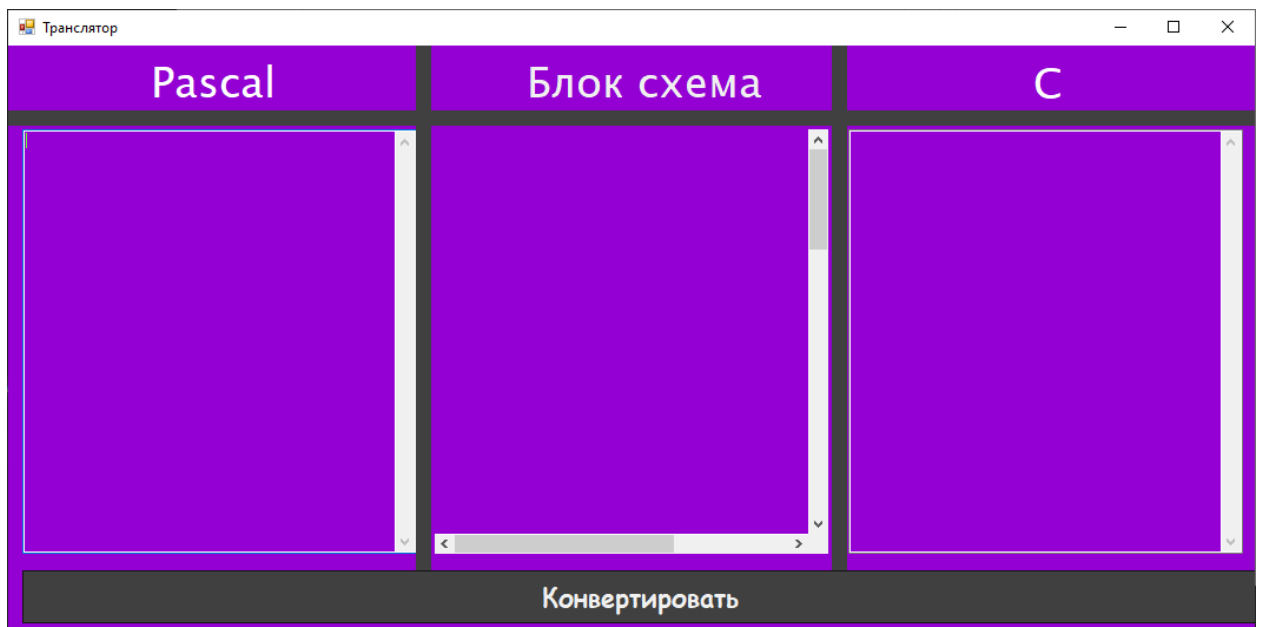


Рисунок 2 – Интерфейс программы

Левое окно позволяет вводить код на языке Pascal, по середине рисуется Блок-схема, а правое получить результат конвертации на языке C.

Для того, чтобы программа корректно выполнила конвертацию, при написании кода на языке Pascal следует самостоятельно проверять логику работы программы. Все команды, кроме циклов и условий, должны прописываться по одному, это позволяет расширить функционал работы программы и производить конвертацию отдельных ключевых слов.

После завершения ввода требуется нажать кнопку «Конвертировать», которая запустит программу конвертации.

В данном примере(рис.3) показаны возможности программы, такие как:

1. Реализация ввода/вывода.
2. Реализация цикла for.
3. Работа с массивами (численными и строковыми).

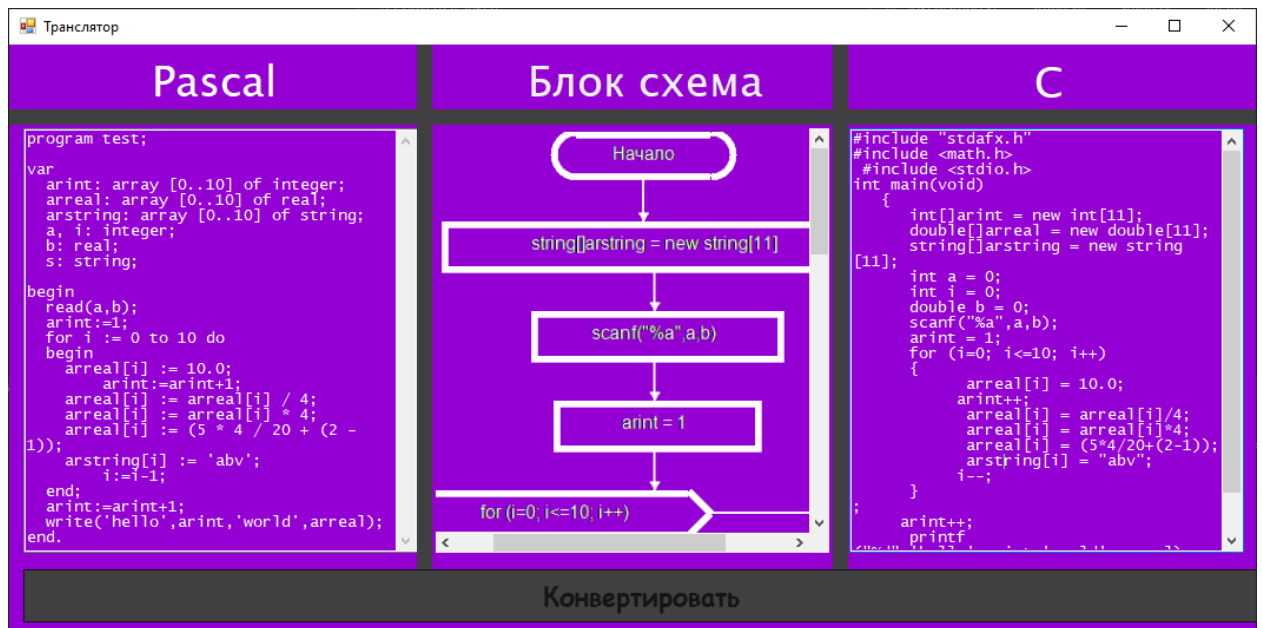


Рисунок 3 – Результат работы программы

Ниже приведены участки кода, показывающие операции конвертации:

Для корректной работы конвертора потребовалось описать соответствие для типов данных:

```
String type;
switch (Semantics.ids[index].type)
{
    case "boolean":
        type = "bool";
        break;
    case "real":
        type = "double";
        break;
    case "integer":
        type = "int";
        break;
    case "string":
        type = "string";
        break;
    default:
        Code.GenerationError = "Замечен недопустимый тип переменной";
}
```

```

return false;
}

```

Для реализации условных операторов реализовано следующее:

```

case "if":
{
    indexT++;
    cplusCode += deep + "if (";
    while (Code.Tokens[indexT].value != "then")
    {
        switch (Code.Tokens[indexT].value)
        {
            case "and":
                cplusCode += "&&";
                break;
            case "or":
                cplusCode += "||";
                break;
            case "<>":
                cplusCode += "!=";
                break;
            case "mod":
                cplusCode += "%";
                break;
            case "div":
                cplusCode += "/";
                break;
            case "=":
                cplusCode += "==";
                break;
            default:
                cplusCode +=
Code.Tokens[indexT].value.Replace("'", "\\");
                break;
        }
        indexT++;
    }
    cplusCode += ") ";
}
break;

```

Таким образом была выстроена работа конвертора, основанная на соответствии двух языков программирования и графической модели.

## **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы были изучены современные методы разработки, основы синтаксического и лексического анализа, а также были получены навыки в области создания конвертора.

Эта цель была достигнута частично путем выполнения следующих задач:

- реализованы модули программного продукта;
- создан интерфейс взаимодействия пользователя и программы;
- отчасти реализована программа конвертации на языке C#;

На основе разработанного алгоритма и полученных знаний приложение можно усовершенствовать.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Карпов, Ю.Г. Теория и технология программирования. Основы построения трансляторов: Учеб. пособие / Ю.Г Карпов Санкт-Петербург: БХВ-Петербург 2005. -272с.
2. Свердлов, С. З. Языки программирования и методы трансляции: учеб. Пособие / С.З Свердлов - Санкт-Петербург : Питер, 2007. - 638 с.
3. Яковлева Л. Л Информатика и программирование/Л. Л Яковлева Учеб. пособие. В 2 ч. Ч. 1. - Чита : ЗабГУ, 2014. - 213 с.
4. Черпаков И. В Основы программирования / И. В Черпаков Учебник и практикум - М. : Издательство Юрайт, 2017. – 219с.
5. Себеста, Роберт У Основные концепции языков программирования / пер. с англ. - 5-е изд. - Москва : Вильямс, 2001. - 672с.
6. Википедия – Свободная энциклопедия [Электронный ресурс] – URL: [https://ru.wikipedia.org/wiki/Си\\_\(язык\\_программирования\)](https://ru.wikipedia.org/wiki/Си_(язык_программирования)) (Дата обращения: 10.12.2021).
7. Википедия – Свободная энциклопедия [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Блок-схема> (Дата обращения: 09.12.2021).



## ПРИЛОЖЕНИЕ

Код файла Syntax.cs (Синтаксический анализатор):

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

namespace Converter
{
    class Syntax
    {
        public Syntax()
        {
            Code.SyntError = "none";
            code = Code.AllCode;
            int line = 1, column = -1;
            for (int i = 0; i < code.Length; i++, column++)
            {
                if (code[i] == '\n')
                {
                    line++;
                    column = -1;
                }

                if (code[i] == "\"")
                {
                    int j = 1;
                    try
                    {
                        while (code[i + (j++)] != "\"");
                    }
                }
            }
        }
    }
}
```

```

        code = code.Remove(++i, j - 2);
    }
    catch (Exception)
    {
        Code.SyntError = "В строке " + line + " столбце " + column + "
ожидалось \' (апостраф, символ конца строки)";
        return;
    }

}

code = code.Replace("\r\n", " ");
while (code.IndexOf(" ") != -1)
    code = code.Replace(" ", " ");
}

public bool GoAnalyze()
{
    if (Code.SyntError != "none")
        return false;

    Grammar.Init();
#if DEBUG
    Grammar.Check();
#endif

    Configuration config = new Configuration();
    Stack<Configuration> history = new Stack<Configuration>();
    history.Push(config);

    while (true)
    {
        if (config.s == 'q')
        {
            if (config.L2.Count != 0)

```

```

{
    if (!config.L2.Peek().term)
    {

        Configuration.elem item = new Configuration.elem(config.L2.Pop());
        Rule current = Grammar.p.Find(x => x.leftSide == item.value);
        if (item.value != "string") //прверка не ожидалась ли строка
        {
            if (item.value != "chislo" && item.value != "integer" && item.value
!= "real")

                {
                    if (item.value != "id")
                    {

                        config.L1.Push(new Configuration.elem(current.leftSide, false,
1));

                        for (int altIndex = current.rightSide[0].Count - 1; altIndex >= 0;
altIndex--)

                            {
                                if (current.rightSide[0][altIndex].notTerminal != "")
                                {
                                    config.L2.Push(new
Configuration.elem(current.rightSide[0][altIndex].notTerminal, false, 1));
                                }
                                else
                                {
                                    config.L2.Push(new
Configuration.elem(current.rightSide[0][altIndex].terminal, true, 1));
                                }
                            }
                        }
                    else
                    {
                        if (Code.Tokens[config.i].klass == "идентификатор")
                        {

```

```

        config.L1.Push(new Configuration.elem(current.leftSide,
false, 1));

        config.L2.Push(new
Configuration.elem(Code.Tokens[config.i].value, true, 1));

    }
    else
    {
        config.L1.Push(new Configuration.elem(current.leftSide,
false, 1));

        for (int altIndex = current.rightSide[0].Count - 1; altIndex >=
0; altIndex--)

            {
                if (current.rightSide[0][altIndex].notTerminal != "")
                {
                    config.L2.Push(new
Configuration.elem(current.rightSide[0][altIndex].notTerminal, false, 1));
                }
                else
                {
                    config.L2.Push(new
Configuration.elem(current.rightSide[0][altIndex].terminal, true, 1));
                }
            }
    }
}
else
{
    if (Code.Tokens[config.i].klass == "число ")
    {
        if (Code.Tokens[config.i].type == "real")
        {
            config.L1.Push(new Configuration.elem(current.leftSide,
false, 2));

```

```

        config.L2.Push(new
Configuration.elem(Code.Tokens[config.i].value, true, 2));
    }
    else
    {
        config.L1.Push(new Configuration.elem(current.leftSide, false, 1));
        config.L2.Push(new Configuration.elem(Code.Tokens[config.i].value, true, 1));
    }
}
else
{
    config.L1.Push(new Configuration.elem(current.leftSide, false, 1));
    for (int altIndex = current.rightSide[0].Count - 1; altIndex >= 0; altIndex--)
    {
        if (current.rightSide[0][altIndex].notTerminal != "")
        {
            config.L2.Push(new Configuration.elem(current.rightSide[0][altIndex].notTerminal, false, 1));
        }
        else
        {
            config.L2.Push(new Configuration.elem(current.rightSide[0][altIndex].terminal, true, 1));
        }
    }
}
}
else
{
    if (Code.Tokens[config.i].klass == "српoкa")
    {
        config.L1.Push(new Configuration.elem(current.leftSide, false, 1));
        config.L2.Push(new Configuration.elem(Code.Tokens[config.i].value, true, 1));
    }
    else

```

```

        {
            config.L1.Push(new Configuration.elem(current.leftSide, false, 1));
            for (int altIndex = current.rightSide[0].Count - 1; altIndex >= 0; altIndex--)
            {
                if (current.rightSide[0][altIndex].notTerminal != "")
                {
                    config.L2.Push(new Configuration.elem(current.rightSide[0][altIndex].notTerminal, false, 1));
                }
                else
                {
                    config.L2.Push(new Configuration.elem(current.rightSide[0][altIndex].terminal, true, 1));
                }
            }
        }
        history.Push(config);
    }
    else
    {
        if (config.i < Code.Tokens.Count)
        {
            Configuration.elem item = new Configuration.elem(config.L2.Peek());
            if ((Code.Tokens[config.i].value == item.value) || (item.value == "lambda"))
            {
                config.L2.Pop();
                config.L1.Push(item);
                if (item.value != "lambda") config.i++;
            }
            else
            {
                config.s = 'b';
            }
        }
        else

```

```

        {
            Code.SyntError = "Неожиданный конец файла";
            return false;
        }
        history.Push(config);
    }
}
else
{
    if (config.i == Code.Tokens.Count)
    {
        config.s = 't';
        config.L2.Push(new Configuration.elem("lambda", true, 0));
        history.Push(config);
        Code.conclusion = config.Seq();
        return true;
    }
    else
    {
        ErrorForm(warn);
        return false;
    }
}
else
{
    if (!config.L1.Peek().term)
    {
        if (!config.L2.Peek().term || !(config.L1.Peek().value == "id"))
        {
            Configuration.elem item = new Configuration.elem(config.L1.Pop());
            Rule current = Grammar.p.Find(x => x.leftSide == item.value);
            if (current.rightSide.Count > item.altIndex)
            {

```

```

for (int elemIndex = 0; elemIndex < current.rightSide[item.altIndex - 1].Count; elemIndex++)
{
    if (config.L2.Count != 0)
    {
        config.L2.Pop();
    }
    else
    {
        return false;
    }
}

for (int elemIndex = current.rightSide[item.altIndex].Count - 1; elemIndex >= 0;
elemIndex--)
{
    if (current.rightSide[item.altIndex][elemIndex].notTerminal != "")
    {
        config.L2.Push(new
        Configuration.elem(current.rightSide[item.altIndex][elemIndex].notTerminal,      false,
        item.altIndex + 1));
    }
    else
    {
        config.L2.Push(new
        Configuration.elem(current.rightSide[item.altIndex][elemIndex].terminal,      true,
        item.altIndex + 1));
    }
}
item.altIndex++;
config.L1.Push(item);
config.s = 'q';
}
else
{
    if (config.i == 0 && item.value == "main")

```



```

        {
            return false;
        }
        else
        {
for (int elemIndex = 0; elemIndex < current.rightSide[item.altIndex - 1].Count;
elemIndex++)
        {
            config.L2.Pop();
        }
        config.L2.Push(new Configuration.elem(item.value, false, 1));
        }
    }
    else
    {
        config.L2.Pop();
        config.L2.Push(new Configuration.elem(config.L1.Pop().value, false, 1));
    }

    }

    else
    {
        Configuration.elem item = new Configuration.elem(config.L1.Pop());
        config.L2.Push(item);
        if (item.value != "lambda") config.i--;
    }
    history.Push(config);
}
}
}
private String code;
}
}

```