Муромский институт (филиал) федерального государственного бюджетного образовательного учреждения высшего образования «Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых»

Факультет Информационных технологий
Кафедра Программной инженерии

КУРСОВАЯ РАБОТА

по Теории алгоритмов и формальных языков
(наименование дисциплины)

Тема Транслятор с подмножества языка Pascal

Руководитель

_____ Кульков Я. Ю.
(оценка) (фамилия, инициалы)

_____
(подпись) (дата)

Члены комиссии

Студент ПИн-119
(группа)

_____ Мартынов Е.С.
(подпись) (Ф.И.О.) (фамилия, инициалы)

_____ _____
(подпись) (Ф.И.О.) (подпись) (дата)

Муром 2021

В данной курсовой работе разработан транслятор с подмножества языка Pascal. В ходе выполнения курсовой работы произведен анализ и сбор требований к разрабатываемой программе. При разработке применены язык C#, среда разработки Visual Studio и технологии Windows Forms. Приложение протестировано и полностью работоспособно. Полученный программный продукт можно применять по его прямому назначению.

In this course work, the translator from a subset of the Pascal language has been developed. In the process of creating a course work, the collection and analysis of the requirements for the projected program was made. The development used the C# language, the Visual Studio development environment and the Windows Forms technology. The program has been tested and fully functional. The resulting product can be used for its intended purpose.

Содержание

Введение

Начиная с XX века, компьютеры становились всё более и более заметной частью нашей жизни. Впервые появившись, как громоздкие калькуляторы, они проделали путь до незаменимых компаньонов, окружающих каждого человека ежесекундно. Но, как при своём зарождении, так и сейчас, для обеспечения их функционирования необходимо переводить понятные любому человеку инструкции в точно такие же, но уже понятные для машины. Для этого были разработаны языки программирования. Однако не любая написанная программа может быть выполнена. Например, она может содержать в себе ошибки, что в итоге приведёт либо к неверному результату работы, либо же вовсе к её неисполнению. Избежать этого помогают специальные программы – трансляторы, которые, помимо прочих своих функций, заранее предупреждают программиста о вышеописанных проблемах.

Темой курсовой работы является разработка транслятора с подмножества языка Pascal.

Достаточной будет реализация шагов, связанных с интерпретацией кода, а именно лексического и синтаксического анализа, а также метода разбора сложных выражений (как элемента синтаксического анализа) и части семантического анализа. Для демонстрации корректности функционирования разработанных алгоритмов берётся не весь язык Pascal, а лишь его подмножество.

Таким образом, целью курсовой работы является создание приложения, занимающегося разбором и анализом переданного ей программного кода на подмножестве языка Pascal.

Главной задачей будет качественная реализация всех пунктов задания.

1 Анализ технического задания

Так как целью данной курсовой работы является создание транслятора с подмножества языка Pascal, разрабатываемое приложение должно содержать:

– реализацию лексического анализа;

– реализацию синтаксического анализа на основе LR(k)-грамматик;

– реализацию метода Бауэра-Замельзона для разбора сложных выражений;

– реализацию части семантического анализа;

– развёрнутую диагностику ошибок.

Требования к используемому подмножеству языка Pascal:

– у идентификатора 8 символов значащие;

– не менее 3-х директив описания переменных;

– сложный арифметический оператор;

– простое логическое выражение;

– оператор цикла for … to … do.

Для удобства ввода и получения информации приложению стоит реализовать графический интерфейс. Результаты всех анализов можно будет отображать в табличной форме, что сделает их понятнее. Кроме того, благодаря этому можно реализовать как ввод текста программы, так и загрузку её из текстового файла.

Для реализации синтаксического анализа и метода Бауэра-Замельзона понадобится использовать такой тип структуры данных, как стек. Его можно реализовать и самостоятельно, но лучшим решением будет использовать реализацию, уже присутствующую в стандартной библиотеке языка.

При создании функционала стоит учесть, что при завершении одного из анализов с ошибкой, запуск остальных не должен быть осуществлён, так как все они полагаются на успешный результат всех предыдущих шагов.

В результате мы приходим к выводу, что лучшим вариантом для разработки будет являться язык C# с использованием технологии Windows Forms. Приложение будет разработано для ОС Windows 10, как самой распространённой в данный момент времени.

2 Описание грамматики языка

Для работы синтаксического анализатора необходимо разработать грамматику для заданного языка. Она должна состоять из множества терминалов, нетерминалов и правил; отдельно указывается имя первого правила.

Результат разработки грамматики для заданного подмножества языка Pascal:

G = <T,N,P,Prog>

T = {var, integer, real, float, begin, end, for, to, do, downto, :, ;, >, <, =, <>, <=, >=, :=, ., ,, +, -, *, /, (,)}

N= {< Prog >,< список описания >,< описание >,< список переменных >,< тип >,< список операторов >,< оператор >,< присваивание в цикле >,< присваивание >,< блок операторов >,< операнд >,< цикл >}

P= {

<Prog> ::= var <список описания> begin <список операторов> end.

<список описания>::=<описание> | <список описания><описание>

<описание>::=<список переменных>:<тип>;

<список переменных>::=id|<список переменных>,id

<тип>::= integer|real|float

<список операторов>::=<оператор>|<список операторов><оператор>

<оператор>::=<присваивание>|<цикл>

<присваивание в цикле>::= Id:=expr

<присваивание>::= <присваивание в цикле>;

<блок операторов>::=<оператор>|begin<список операторов>end;

<операнд>::=id|lit

<цикл>::= for <присваивание в цикле> to <операнд> do <блок операторов> | for <присваивание в цикле> downto <операнд> do <блок операторов>

}

Полученная грамматика была проверена, результат проверки приведён ниже:
var a,b: integer; begin for i:=10 to 100 do begin c := b; a : = a + 10; end; end.

var <список описания> begin <список операторов> end.

var <описание> | <список описания><описание> begin <оператор>|<список операторов><оператор> end.

var <описание> begin <оператор> end.

var <список переменных>:<тип>; begin <присваивание>|<цикл> end.

var id|<список переменных>,id:<тип>; begin <цикл> end.

var id,id: integer; begin for <присваивание в цикле> to <операнд> do <блок операторов> end.

var id,id: integer; begin for Id:=expr to lit do begin<список операторов>end; end.

var id,id: integer; begin for Id:=expr to lit do begin <оператор><оператор> end; end.

var id,id: integer; begin for Id:=expr to lit do begin <присваивание><присваивание> end; end.

var id,id: integer; begin for Id:=expr to lit do begin Id:=expr; Id:=expr; end; end.

var a,b: integer; begin for i:=10 to 100 do begin c := b; a : = a + 10; end; end.

В программе будет использоваться реализация синтаксического анализа на основе LR(k)-грамматик. Для его реализации необходимо составить граф состояний автомата и решающую таблицу детерминированного автомата. Они приведены в таблицах 1 и 2 в Приложении 3.

По результату построения таблицы 2 из Приложения 3 видно, что, так как для некоторых состояний не необходимо использовать символы не просмотренной части входной цепочки, грамматика является LR(0).

3 Разработка архитектуры системы и алгоритмов

Разрабатываемая система будет состоять из главного окна, включающего в себя весь функционал, и из отдельных классов для разбора сложных арифметических выражений и остальных этапов анализа программы.

Взаимодействие пользователя с программой будет производиться посредством клавиатуры и мыши.

3.1 Алгоритм лексического анализа

Алгоритм лексического анализа, разрабатываемый для приложения, принимает на массив строк, представляющий текст программы, и в ходе своей работы он заполняет списки лексем, переменных и чисел в классе. Далее их можно получить через специальные методы.

Сам алгоритм представляет собой цикл, который проходится по всему переданному массиву и производит проверки на соответствие текущего символа какому-либо типу. При совпадении, программа получает все символы данного типа вплоть до поступления символов другого типа. Если же тип неизвестен, то выводится ошибка о неподдерживаемом символе. Кроме того, поддерживается комментирование строк – в таком случае игнорируется вся строка вплоть до её конца.

3.2 Алгоритм синтаксического анализа

Алгоритм синтаксического анализа, разрабатываемый для приложения, использует списки, заполненные в результате лексического анализа, полученный в результате лексического анализа, в котором все имена переменных и числа заменены на определённые терминальные символы. Алгоритм выдаёт информацию о результате своей работы (например, удачно завершена или завершена из-за ошибки), а также заносит результат работы подпрограммы разбора выражений по

методу Бауэра-Замельзона в специальный список в классе. Алгоритм построен на основе LR(k)-грамматик. При разработке использовалась решающая таблица детерминированного автомата, содержащая в себе правила обработки различных поступающих лексем, распределённые по так называемым «состояниям».

После запуска метода запускается цикл, в котором проверяется текущее внутреннее состояния метода и, в зависимости от результата первой проверки, текущий элемент (взятый с вершины стека) проверяется на допустимость. Если элемент допустим, то выполняется предписанное ему действие (сдвиг, переход или приведение), после чего цикл отправляется на новый виток. В противном случае, работа метода завершается, а на экран выводится ошибка с ожидаемой и встретившейся лексемой.

3.3 Алгоритм разбора сложных выражений по методу Бауэра-Замельзона

Частью алгоритма синтаксического анализа является алгоритм разбора сложных выражений по методу Бауэра-Замельзона. Он расположен в отдельном классе программы , а единственный публичный метод принимает на вход строку – арифметическое выражение. Возвратом данного метода является либо успешная проверка выражения, либо ошибка в случае неверного выражения

4 Методика испытаний

Для обеспечения гарантии работоспособности приложения, должно быть произведено тестирование всех использованных алгоритмов. Результаты тестирования приведены в таблице 3.

Таблица 1 - Результаты тестирования

Программа	Вывод	Пояснение

`var a,b: integer;`
`begin`
`  for i := 10 to 100 do begin`
`  c := b + 4;`
`  a := a;`
`  end;`
`end.`
	Анализ успешно завершён	Ни один алгоритм не нашёл в данной программе ошибок

`var a,b,Ф: integer;`
`begin`
`  for i := 10 to 100 do begin`
`  c := b + 4;`
`  a := a;`
`  end;`
`end.`
	Недопустимый символ - ф	Лексический анализатор выдал данную ошибку, так как кириллические символы в коде не поддерживаются

`var a,b,DarthVader: integer;`
`begin`
`  for i := 10 to 100 do begin`
`  c := b + 4;`
`  a := a;`
`  end;`
`end.`
	Слишком большой идентификатор - darthvader	Лексический анализатор выдал данную ошибку, так как имя переменной «DarthVader» состоит из 10 символов

Таблица 1 (продолжение)

Программа	Вывод	Пояснение

var a,b: integer;
begin do
  for i := 10 to
100 do begin
  c := b + 4;
  a := a;
  end;
end.
	Ожидалось «for или идентификатор» а встретилось «do»	Синтаксический анализатор выдал ошибку

var a,b: integer;
begin
  for i := 10 to
100 do begin
  c := b +- 4;
  a := a;
  end;
end.
	Ошибка при разборе арифметического выражения	Метод Бауэра-Замельзона выдал данную ошибку так как +- - недопустимый оператор

Результаты, полученные в ходе тестирования, позволяют сделать заключение о том, что разработанная программа соответствует требованиям технического задания.

5 Руководство пользователя

Минимальные системные требования:

1. Операционная система Windows 10;
2. Процессор с частотой 1 ГГц;
3. 1 ГБ оперативной памяти;
4. 1 МБ места на жёстком диске;
5. Необходимо наличие монитора с разрешением не менее 1280x720 и устройств ввода (клавиатуры и мыши).

Программа не требует установки. Для начала работы с приложением необходимо запустить исполняемый файл CodeAnalysis.exe. На рисунке 1 показан внешний вид приложений после запуска.

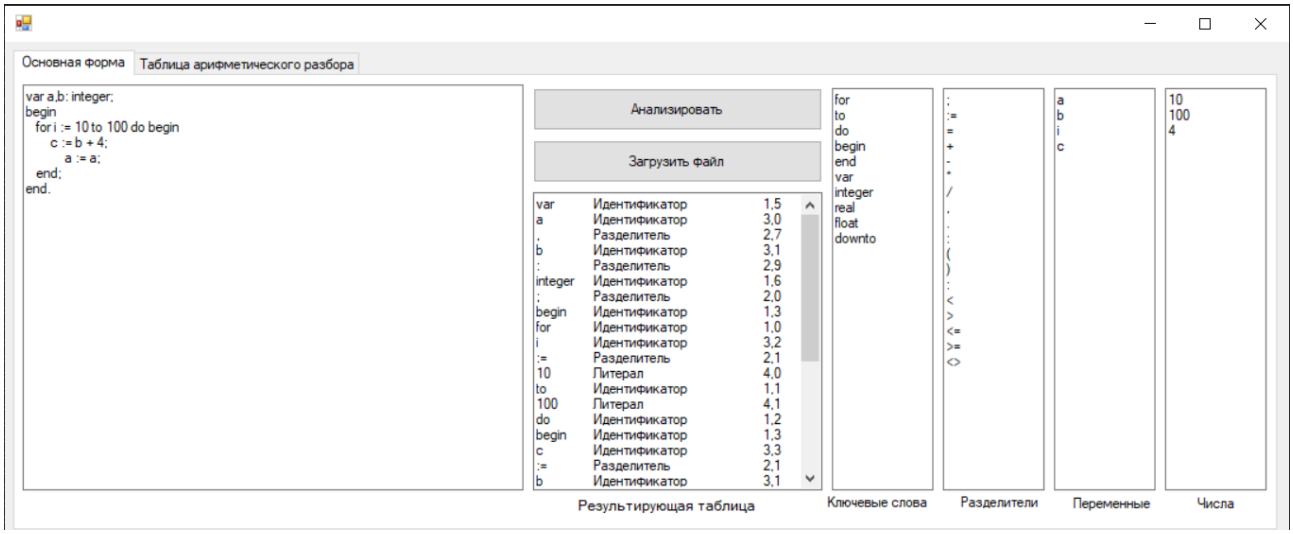

Рисунок 1 - Запущенное приложение

Для начала анализа программы необходимо прочитать информацию из текстового файла при помощи кнопки «Загрузить файл» или же ввести код в текстовое поле. После этого необходимо нажать кнопку «Анализировать».

Посредством переключения вкладок можно просмотреть результаты проведения арифметического анализа.

При возникновении ошибок программа покажет сообщение об ошибке, после чего остановит работу анализаторов; последующие анализы не будут произведены.

6 Руководство программиста

Приложение создано на языке C# с использованием программной платформы .NET Framework 4.7.2 и технологии Windows Forms.

При разработке было создано 4 класса, которые будут описаны ниже; также присутствует базовый класс Program, осуществляющий запуск программы (создан автоматически).

6.1 Класс MainForm

Класс MainWindow является классом Windows Forms и содержит в себе определение окна приложения.

Методы класса MainWindow:

1. public MainForm() – конструктор класса;
2. private void Analise() – выполнение всех видов анализа и вывод данных на форму в случае успешного завершения анализа;
3. private void buttonAnalise_Click(object sender, EventArgs e) – обработчик события, возникающего при нажатии на кнопку «Анализировать». Очищает текстовые поля и проводит анализ;
4. private void Print(TextBox textBox, List<string> list) – функция, выводящая переданный список в переданный TextBox в формате одного элемента списка на строчку;
5. private void ClearTextFields() – функция, очищающая текстовые поля на форме;
6. private void buttonLoad_Click(object sender, EventArgs e) – обработчик события, возникающего при нажатии на кнопку «Загрузить файл». Считывает содержимое файла и выводит его в текстовое поле на форме;

6.2 Класс Translator

Класс Translator содержит в себе алгоритмы проведения лексического и синтаксического анализа.

Поля класса Translator:

1. private enum _type – поле, в котором хранятся возможные типы лексем;
2. private static string _buf – поле, временно сохраняющее последовательно расположенные символы введённого программного кода;
3. private readonly static List<string> _keyWords – список, содержащий все возможные ключевые слова для данной программы
4. private readonly static List<string> _separators – список, содержащий все возможные разделители для данной программы.
5. private static List<string> _variables – список, в который будут записаны все переменные в ходе лексического анализа
6. private static List<string> _numbers – список, в который будут записаны все обнаруженные числа в ходе лексического анализа
7. private static List<string> _links - список в который будут записаны ссылки между списком лексем и списками с ключевыми словами, переменные, числами и разделителями.
8. private static List<string> _bauerOuput – список, содержащий вывод алгоритма разбора арифметических выражений
9. private static List<string> _lexicList – список лексем, полученных в результате лексического анализа
10. private static List<_type> _typeList – список типов лексем
11. private readonly static List<char> FSP – список первых элементов символьных пар
12. private readonly static List<char> SSP – список вторых элементов символьных пар
13. private static Stack<int> _state – стек состояний, использующийся при синтаксическом анализе
14. private static Stack<string> _program – стек лексем, использующийся при синтаксическом анализе

15. private static int _shiftCounter – счётчик сдвигов, использующийся при синтаксическом анализе

Методы класса Translator:

1. public static void LexicalAnalise(string[] input) – метод, принимающий в себя массив строк, представляющий программу и осуществляющий лексический анализ;
2. private static void StringLexicalAnalise(string input) – метод, осуществляющий лексический анализ одной строки
3. private static void LexicalClassification() – метод, производящий лексическую классификацию
4. public static void SyntaxAnalise() – метод, осуществляющий синтаксический анализ
5. private static void Shift() – метод, осуществляющий сдвиг при синтаксическом анализе
6. private static void Fold(int count, string item) – метод, осуществляющий свёртку при синтаксическом анализе
7. private static void ComplexMathOperation() – метод, осуществляющий разбор математического с помощью специального класса
8. private static void State0()-State42() – методы-состояния при синтаксическом анализе
9. public static void ClearLists() – метод, очищающий все списки в классе
10. public static List<string> GetLexicList() – метод, возвращающий список лексем
11. public static List<string> GetTypeList() – метод, возвращающий список типов
12. public static List<string> GetVariablesList() – метод, возвращающий список переменных
13. public static List<string> GetNumbersList() – метод, возвращающий список чисел

14. public static List<string> GetLinksList() - – метод, возвращающий список ссылок

15. public static List<string> GetKeyWordsList() - – метод, возвращающий список ключевых слов

16. public static List<string> GetSeparatorsList() - – метод, возвращающий список разделителей

17. public static List<string> GetBauerOutput() - – метод, возвращающий вывод метода арифметического разбора

6.3 Класс Bauer

Класс Bauer содержит в себе алгоритм проведения синтаксического анализа.

Поля класса Bauer:

1. private Stack<string> Operands – стек, содержащий операнды;
2. private Stack<char> _functions – стек, содержащий операции
3. private int _operationCounter – счётчик операций
4. public List<string> Output { get; private set; } – вывод алгоритма

Методы класса Bauer:

1. public bool Calc(string s) – разбор выражения
2. private void PopFunction() – вычисление операции
3. private bool CanPop(char operation, Stack<char> Functions) – можно ли взять следующую операцию
4. private object GetToken(string s, ref int position) – получение следующего элемента
5. private char ReadFunction(string s, ref int position) – получение функции
6. private string ReadOperand(string s, ref int position) – получение операнда
7. private void ReadWhiteSpace(string s, ref int position) – считывание пробелов

6.4 Класс AnaliseException

Данный класс является наследником от класса Exception и содержит два конструктора:

public AnaliseException(string expected, string was) : base($"Ожидалось: \"{expected}\" а встретилось {was}") { }

public AnaliseException(string message) : base(message) { }

Заключение

В ходе данной курсовой работы была выполнена разработка транслятора с подмножества языка Pascal.

В ходе выполнения были решены следующие задачи:

– реализация лексического анализа;

– реализация синтаксического анализа на основе LR(k)-грамматик;

– реализация метода Бауэра-Замельзона для разбора сложных выражений;

– реализация части семантического анализа;

– наличие развёрнутой диагностики ошибок.

Разработанная программа может быть усовершенствована.

Таким образом, в курсовой работе были реализованы все пункты технического задания.

Список используемой литературы

1. Pascal Case Statement: сайт. – URL: https://www.tutorialspoint.com/pascal/pascal_case_statement.htm (дата обращения: 13.09.2021). – Текст: электронный.

2. Язык программирования C# и платформа .NET: сайт. – URL: https://metanit.com/sharp/ (дата обращения: 25.10.2021). – Текст: электронный.

3. Stack Overflow: сайт. – URL: https://stackoverflow.com/ (дата обращения: 17.11.2021). – Текст: электронный.

Приложение 2 – Снимки окон программы (скриншоты программы)

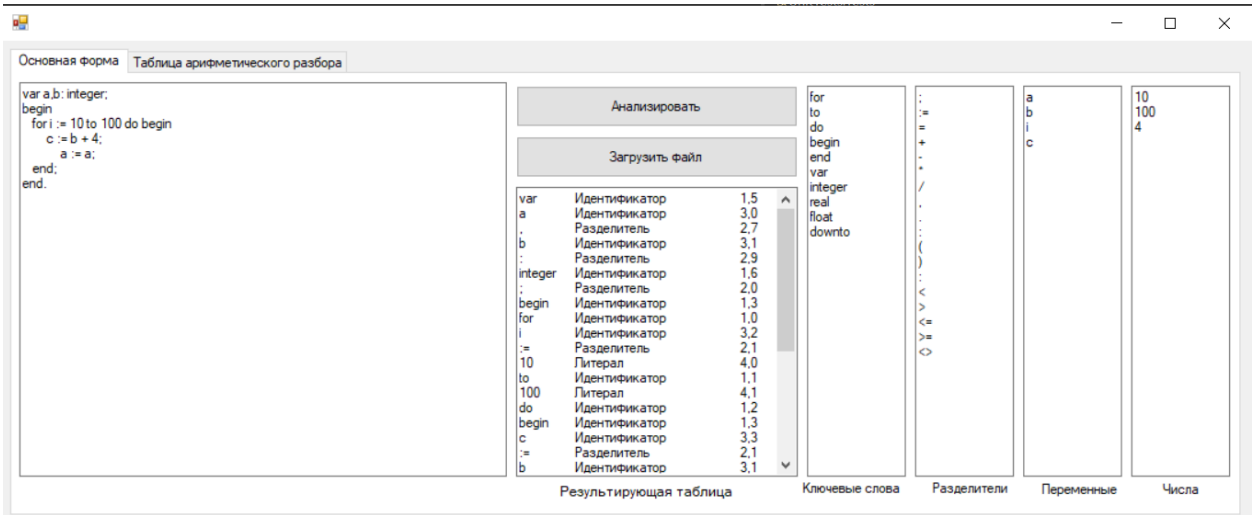

Рисунок 2 – скриншот работы программы

Рисунок 3 – скриншот работы программы

Приложение 3. Решающие таблицы

Таблица 1 – Решающая таблица

Состояние	Предыдущее состояние	Правила	Переход
0	-	<S>::=•<prog>$ <prog>::=•var<список опис.>begin<список операторов>end	1 2
1	0	<S>::=<prog>•$	x
2	0	<prog>::=var•<список опис.> begin<список операторов>end <список опис.>::= •<описание> <список опис.>::= •<список описания>< описание > <описание>::=•<список переменных><тип>; <список переменных>::=•id <список переменных>::=•<список переменных>,id	3 4 3 5 6 5
3	2	<prog>::=var<список опис.> • begin<список операторов>end <список опис.>::= <список описания>•< описание > <описание>::=•<список переменных><тип>; <список переменных>::=•id <список переменных>::=•<список переменных>,id	8 4 5 6 5
4	2,3	<список опис.>::= <описание>•	x
5	2,3	<список переменных>::=<список переменных>•,id <описание>::=<список переменных>•<тип>; <тип>::= •integer <тип>::=•real <тип>::=•float	13 9 10 11 12
6	2,3	<список переменных>::=id•	x
7	13	<список переменных>::=<список переменных>,id•	x
8	3	<prog>::=var<список опис.> begin•<список операторов>end <список операторов>::=•<оператор> <список операторов>::=•<список операторов><оператор> <оператор>::=•<присваивание> <присваивание>::= •<присваивание в цикле>; <оператор>::=•<цикл> <цикл>::= •for <присваивание в цикле> to <операнд> do <блок операторов> <цикл>::= •for <присваивание в цикле> downto <операнд> do <блок операторов> <присваивание в цикле>::=• Id:=expr	14 15 14 17 18 19 20 20 21
9	5	<описание>::=<список переменных><тип>•;	22
10	5	<тип>::= integer•	x
11	5	<тип>::= real•	x
12	5	<тип>::= float•	x
13	5	<список переменных>::=<список переменных>,•id	7
14	8,34	<prog>::=var<список опис.> begin<список операторов>•end <список операторов>::=<список операторов>•<оператор> <оператор>::=•<присваивание>	23 38 17 18 19

		<присваивание>::= •<присваивание в цикле>; <оператор>::=•<цикл> <цикл>::= •for <присваивание в цикле> to <операнд> do <блок операторов> <цикл>::= •for <присваивание в цикле> downto <операнд> do <блок операторов> <присваивание в цикле>::=• Id:=expr	20 20 21
15	8, 34	<список операторов>::=<оператор>•	x
17	8,14, 32, 34	<оператор>::=<присваивание>•	x
18	8,14,32,34	<присваивание>::= <присваивание в цикле>•;	35
19	8,14, 32,34	<оператор>::=<цикл>•	x
20	8,14,32,34	<цикл>::= for •<присваивание в цикле> to <операнд> do <блок операторов> <цикл>::= for •<присваивание в цикле> downto <операнд> do <блок операторов> <присваивание в цикле>::=• Id:=expr	24 24 21
21	8,14,20,32,34	<присваивание в цикле>::= Id•:=expr	25
22	9	<описание>::=<список переменных><тип>;•	X
23	14	<prog>::=var<список опис.> begin<список операторов>end•	X
24	20	<цикл>::= for <присваивание в цикле>• to <операнд> do <блок операторов> <цикл>::= for <присваивание в цикле>• downto <операнд> do <блок операторов>	27 28
25	21	<присваивание в цикле>::= Id:= •expr	26
26	25	<присваивание в цикле>::= Id:= expr•	x
27	24	<цикл>::= for <присваивание в цикле> to •<операнд> do <блок операторов> <операнд>::=•id <операнд>::=•lit	29 30 31
28	24	<цикл>::= for <присваивание в цикле> downto• <операнд> do <блок операторов> <операнд>::=•id <операнд>::=•lit	29 30 31
29	27,28	<цикл>::= for <присваивание в цикле> to <операнд>• do <блок операторов> <цикл>::= for <присваивание в цикле> downto <операнд>• do <блок операторов>	32 32
30	27,28	<операнд>::=id•	x
31	27,28	<операнд>::=lit•	x
32	29	<цикл>::= for <присваивание в цикле> to <операнд> do• <блок операторов> <цикл>::= for <присваивание в цикле> downto <операнд> do• <блок операторов> <блок операторов>::=•<оператор> <оператор>::=•<цикл> <оператор>::=•<присваивание> <присваивание>::= •<присваивание в цикле>; <присваивание в цикле>::=• Id:=expr <цикл>::= •for <присваивание в цикле> to <операнд> do <блок операторов> <цикл>::= •for <присваивание в цикле> downto <операнд> do <блок операторов>	33 33 39 19 17 18 21 20 20

		<блок операторов>::=•begin<список операторов>end;	34
33	32	<цикл>::= for <присваивание в цикле> to <операнд> do <блок операторов>• <цикл>::= for <присваивание в цикле> downto <операнд> do <блок операторов>•	X x
34	32	<блок операторов>::=begin•<список операторов>end; <список операторов>::=•<оператор> <список операторов>::=•<список операторов><оператор> <оператор>::=•<присваивание> <присваивание>::= •<присваивание в цикле>; <оператор>::=•<цикл> <цикл>::= •for <присваивание в цикле> to <операнд> do <блок операторов> <цикл>::= •for <присваивание в цикле> downto <операнд> do <блок операторов> <присваивание в цикле>::=• Id:=expr	36 15 36 17 18 19 20 20 21
35	18	<присваивание>::= <присваивание в цикле>;•	x
36	34	<блок операторов>::=begin<список операторов>•end; <список операторов>::=<список операторов>•<оператор> <оператор>::=•<присваивание> <присваивание>::= •<присваивание в цикле>; <присваивание в цикле>::=• Id:=expr	37 38 17 18 21
37	36	<блок операторов>::=begin<список операторов>end•;	40
38	14	<список операторов>::=<список операторов><оператор>•	x
39	32	<блок операторов>::=<оператор>•	x
40	37	<блок операторов>::=begin<список операторов>end;•	x

Таблица 2 – Решающая таблица

Состояние	Стек	Вход	Действие
0	<S> <prog> var		End S1 S2
1	<prog>		Приведение(-1, <S>)
2	Var <список опис.> <Описание> <Список переменных> id		Сдвиг S3 S4 S5 S6
3	var<список опис.> begin <Описание> <Список переменных> id		Сдвиг S8 S4 S5 S6
4	<описание>		Приведение(-1, <список описания>)
5	<список переменных> , <тип>		Сдвиг S13 S9

	Integer Real float		S10 S11 S12
6	id		Приведение(-1, <список переменных>)
7	<список переменных>,id		Приведение(-3, <список переменных>)
8	var<список опис.>begin <список операторов> <оператор> <присваивание> <присваивание в цикле> <цикл> For id		Сдвиг S14 S15 S17 S18 S19 S20 S21
9	<список переменных><тип> ;		Сдвиг S22
10	integer		Приведение(-1, <тип>)
11	real		Приведение(-1, <тип>)
12	float		Приведение(-1, <тип>)
13	<список переменных>, id		Сдвиг S7
14	var<список опис.> begin<список операторов> <список операторов> <присваивание> <присваивание в цикле> <цикл> For id <оператор>		Сдвиг Сдвиг S17 S18 S19 S20 S21 S38
15	<оператор>		Приведение(-1, <список операторов>)
17	<присваивание>		Приведение(-1, <оператор>)
18	<присваивание в цикле> ;		Сдвиг S35
19	<цикл>		Приведение(-1, <оператор>)
20	For <присваивание в цикле> id		Сдвиг S24 S21
21	Id :=		Сдвиг S25
22	<список переменных><тип>;		Приведение(-3,<описание>)
23	var<список опис.> begin<список операторов>end		Приведение(-5,<prog>)
24	for <присваивание в цикле> to downto		Сдвиг S27 S28
25	Id:= expr		Сдвиг S26
26	Id:=expr		Приведение(-3,<присваивание в цикле>)
27	for <присваивание в цикле> to		Сдвиг

	<операнд> Id lit		S29 S30 S31
28	for <присваивание в цикле> downto <операнд> Id lit		Сдвиг S29 S30 S31
29	for <присваивание в цикле> to <операнд> for <присваивание в цикле> downto <операнд> do		Сдвиг Сдвиг S32
30	id		Приведение(-1,<операнд>)
31	lit		Приведение(-1,<операнд>)
32	for <присваивание в цикле> to <операнд> do for <присваивание в цикле> downto <операнд> do <блок операторов> <оператор> <цикл> For begin		Сдвиг Сдвиг S33 S39 S19 S20 S34
33	for <присваивание в цикле> to <операнд> do <блок операторов> for <присваивание в цикле> downto <операнд> do <блок операторов>		Приведение(-6, <цикл>) Приведение(-6,<цикл>)
34	Begin <оператор> <список операторов> <присваивание> <присваивание в цикле> <цикл> for Id		Сдвиг S15 S14 S17 S18 S19 S20 S21
35	<присваивание в цикле>;		Приведение(- 2,<присваивание>)
36	begin<список операторов>end ;		Сдвиг S37
37	begin<список операторов>end;		Приведение(-4,<блок операторов>)
38	<список операторов><оператор>		Приведение(-2,<список операторов>)
39	<оператор>		Приведение(-1,<блок операторов>)