МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«КУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет физики, математики, информатики
Кафедра алгебры, геометрии и теории обучения математике

КУРСОВАЯ РАБОТА
по дисциплине
Теория формальных языков и трансляций
на тему: РАЗРАБОТКА КОМПИЛЯТОРА МОДЕЛЬНОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ

Обучающегося 3 курса
очной формы обучения
направления подготовки
02.03.03 Математическое обеспечение
и администрирование
информационных систем
Направленность (профиль)
Проектирование информационных
систем и баз данных
Мусонда Салиму

Руководитель: доцент кафедры алгебры, геометрии и теории обучения математике
Селиванова Ирина Васильевна

Допустить к защите:
_____/_____
«_____»_____20_____г.

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Данная курсовая работа заключается в создании компилятора модельного языка программирования.

Компилятор – это техническое средство, которое может транслировать программы, написанные на языке высокого уровня, в программы на машинном языке или на языке ассемблера. Большинство компиляторов переводят программу с некоторого языка программирования в машинный код, который может быть непосредственно выполнен компьютером [3].

Входной информацией для компилятора является описание алгоритма или программа на некотором языке программирования, а выходной информацией является эквивалентное описание алгоритма на машинно-ориентированном языке.

Актуальность данной темы заключается в том, что в настоящее время существует много языков программирования, но реализация новых задач требует создания языков программирования, способных наиболее эффективно решать их. Поэтому важным является изучение приемов и методов, получение практических навыков разработки компиляторов для некоторых модельных языков программирования.

Целью данной курсовой является выработка навыка разработки компилятора модельного языка программирования, изучение основных принципов построения и функционирования компиляторов, закрепление теоретических знаний, полученных при изучении дисциплины.

Задачи:

1) изучить существующие алгоритмы и методы создания компиляторов;
2) разработать формальную грамматику для языка программирования;
3) разработать алгоритмы лексического, синтаксического и семантического анализа, выделив семантические правила, проверяемые компилятором;

4) рассмотреть алгоритм перевода в ПОЛИЗ;

5) разработать алгоритм перевода в ассемблеровский код исходного кода;

6) разработать и отладить приложение, визуализирующее этапы компиляции.

В данной работе реализован компилятор согласно варианту задания №10, содержание которого представлено ниже.

1. Структура программы:

   <программа>::=«{»{/(<описание>|<оператор>);/}«}»
2. Синтаксис команд описания данных:

   <описание>::= *dim*<идентификатор>{,<идентификатор>}<тип>
3. Синтаксис идентификаторов:

   <идентификатор> ::= <буква> <цифра> <непустая последовательность цифр>
4. Описание типов (в порядке следования: целый, действительный, логический):

   <тип>::=integer|real|boolean
5. Синтаксис составного оператора:

   <составной>::= begin <оператор> { ; <оператор> } end
6. Оператор оператора присваивания:

   <присваивание>::= <идентификатор> ass <выражение>
7. Оператор условного перехода:

   <условный>::=switch<выражение>{case Const1:<оператор>case Const2:<оператор>…case ConstN<оператор>}
8. Синтаксис оператора:

   <фиксированного_цикла>::=for each<присвания> to <выражение>do<оператор>
9. Синтаксис оператора:

   <условного_цикла>::= return < оператор > until (<выражение>)
10. Синтаксис оператора:

<ввода>::= read (<идентификатор> {, <идентификатор> })

11.Синтаксис оператора:

<вывода>::= writeln <выражение> {, <выражение> }

12.Признак начала комментария:

/*

13.Признак конца комментария:

*/

Первый этап разработки компиляторов связан с разработкой грамматики языка программирования.

# 1 РАЗРАБОТКА ГРАММАТИКИ МОДЕЛЬНОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ

Существуют несколько основных метода описания синтаксиса языков программирования: формальные грамматики, формы Бэкуса-Наура и диаграммы Вирта.

## 1.1 Формальные грамматики

Формальные грамматики (ФГ) являются математическим аппаратом, который исследует свойства цепочек символов, порожденных заданным набором правил. Фундаментальным понятием для ФГ является понятие цепочки: последовательности символов из некоторого множества (алфавита) [6].

Формальной грамматикой называется четверка вида:

G - ($V_t$, $V_n$, $P$, $S$), где

$V_t$, - Алфавит терминальных символов (терминалов);

$V_n$ - Алфавит нетерминальных символов (нетерминалов);

P - Множество правил вывода грамматики. Элемент множества P называют правилом вывода;

S - Начальный символ грамматики.

Для обозначения нетерминальных символов будем использовать большие буквы латинского алфавита. Множество нетерминальных символов имеет следующий вид.

Множество нетерминальных символов:

P – программа, D – оператор, $D_1$ – составной оператор, $D_2$ – оператор присваивания, $D_3$ – условный оператор, $D_4$ – оператор фиксированного цикла,

$D_5$ – оператор условного цикла, $D_6$ – оператор ввода, $D_7$ – оператор вывода, B – описания, $B_1$ – описание, S – выражения, $S_1$ – выражения, I – идентификатор, $I_1$ – идентификаторы, T – тип, Z – последовательность букв, $Z_1$ – буква, C – число, $C_1$ – цифра, $C_2$ – последовательность цифр, A– арифметическое выражение, L –логическое выражение, Q – знак арифметического выражения,Q1 – знак логического выражения,

Множество терминальных символов:

<T>:=!, %, $ , begin, var, end , ass, if, then, for each, in, while, do, read, writeln, =,+, -, *,/, >,=,<, {, }, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Используя введенные обозначения, опишем правила выводов, применимые в формальной грамматике.

Правила вывода:

P → *program var* B *begin* D *end*

B → T I

I → $I_1$ | I, $I_1$

$I_1$ → $Z_1C_2$ $Z_1$

$C_2$ → $C_1$| $C_2$ $C_1$

T → *% | ! | $*

D → $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$| $D_8$

$D_1$ →*begin* $D_1$ | $D_1$; D *end*

$D_2$ → $I_1$: ass S

$D_3$ → *if* S *then* D [*else* D]

$D_4$ → *for each* I *in* (Const1,Const2,...) D

$D_5$ → *while* D *do* S

$D_6$ → *read* (I)

$D_7$ → *writeln*(S)

S → $S_1$ | S

$S_1 \rightarrow A|L$

$A \rightarrow I1 \mid C1 \mid A\ Q\ A$

$Q \rightarrow + \mid - \mid * \mid /$

$L \rightarrow L\ Q_1\ L$

$Q_1 \rightarrow > \mid = \mid <$

$C \rightarrow 0|1|2|3|4|5|6|7|8|9$

$Z \rightarrow a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z$

## 1.2 Форма Бэкуса - Наура (БНФ)

БНФ – формальная система описания синтаксиса, в которой одни синтаксические категории последовательно определяются через другие категории. Функционально она используется для точного описания синтаксиса языка [4].

Метаязык, предложенный Бэкусом и Науром, использует следующие обозначения:

– символ «::=» отделяет левую часть правила от правой (читается: «определяется как»);

– нетерминалы обозначаются произвольной символьной строкой, заключенной в угловые скобки «»;

– терминалы - это символы, используемые в описываемом языке;

– правило может определять порождение нескольких альтернативных цепочек, отделяемых друг от друга символом вертикальной черты «|» (читается: «или»).

При записи грамматики в форме Бэкуса-Наура используются следующие типы объектов:

- основные символы (или терминальные символы, в частности, ключевые слова);
- металингвистические переменные (или нетерминальные символы), значениями которых являются цепочки основных символов

описываемого языка. Металингвистические переменные изображаются словами заключенными в угловые скобки.

- металингвистические связки (::=, |)

В данной БНФ используются следующие конструкции:

<программа>::=program var <описание> begin <оператор> {; <оператор>} end.

<описание>::= {<идентификатор> {, <идентификатор> } : <тип> }

<идентификатор>::=<буква><непустая последовательность цифр><буква>

<число>::={/цифра/}

<цифра>::=0|1 |2 |3 |4 |5 |6 |7 |8 |9

<буква>::=a| b| c| d| e| f| g| h| i| j| k| l| m| n| o| p| q| r| s| t| u| v| w| x| y| z

<тип>::= % | ! | $ (в порядке следования: целый, действительный, логический)

<шаг>::=<число>

<оператор>::= <составной>| <присваивания>| <условный>| <фиксированного _ цикла>| <условного _ цикла>|<ввода>| <вывода>

<составной>::= begin <оператор> { ; <оператор> } end

<присваивание>::= <идентификатор> ass <выражение>

<условный>::= if<выражение> then <оператор> [ else <оператор>]

<фиксированного цикла>::= for each<идентификатор> in ( Const1,Const2,...) <оператор>

<условного_цикла>::= while <выражение> do <оператор>

<ввода>::= read <идентификатор> {/ , <идентификатор> /};

<вывода>::= writeln <выражение> {, <выражение> }

<выражение>::=<ариф.выражение> |<лог.выражение>

<ариф.выражение>::=<идентификатор>|<число> |<ариф.выражение><знак ариф.операции><ариф.выражение>

<лог.выражение>::=<идентификатор> | <число> |<лог.выражение><знак лог.операции><лог.выражение>

<знак лог.операции>::= > | = | <

<признак начала комментария>::=/*

<признак начала комментария>::=*/

## 1.3 Описание грамматики на языке синтаксических диаграмм

Разработка языка программирования начинается с определения его синтаксиса. Описать синтаксис можно обычным языком. Но более наглядно его можно представить, изобразив визуально в виде синтаксической диаграммы.

Рассмотрим оставшийся метод описания синтаксиса грамматики для данного модельного языка: синтаксические диаграммы (диаграммы Вирта). Они позволяют визуализировать формы Бэкуса-Наура.

Терминальный символ непосредственно присутствует в словах языка и имеет конкретное значение. Терминальные символы обозначаются овалом, в котором указано его значение.

Нетерминальный символ - сущность языка, например, формула или арифметическое выражение и не имеющая конкретного символьного значения. Нетерминальные символы обозначаются прямоугольником, в котором указано его назначение.

Синтаксическая диаграмма - это схема или некое графическое представление описания нетерминального символа языка.

Элементы соединяются между собой направленными линиями, стрелки указывают порядок следования элементов. Альтернативы в правилах обозначаются ветвлением прямых, итерации – слиянием прямых линий.

На рисунках 1-12 представлено описание синтаксиса модельного языка с помощью диаграмм Вирта.

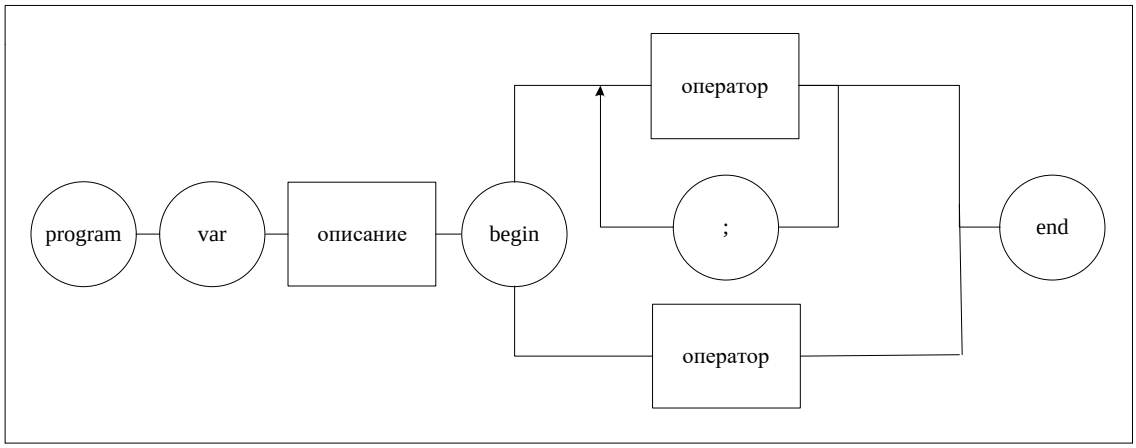


Рисунок 1 - Синтаксическая диаграмма <программа>

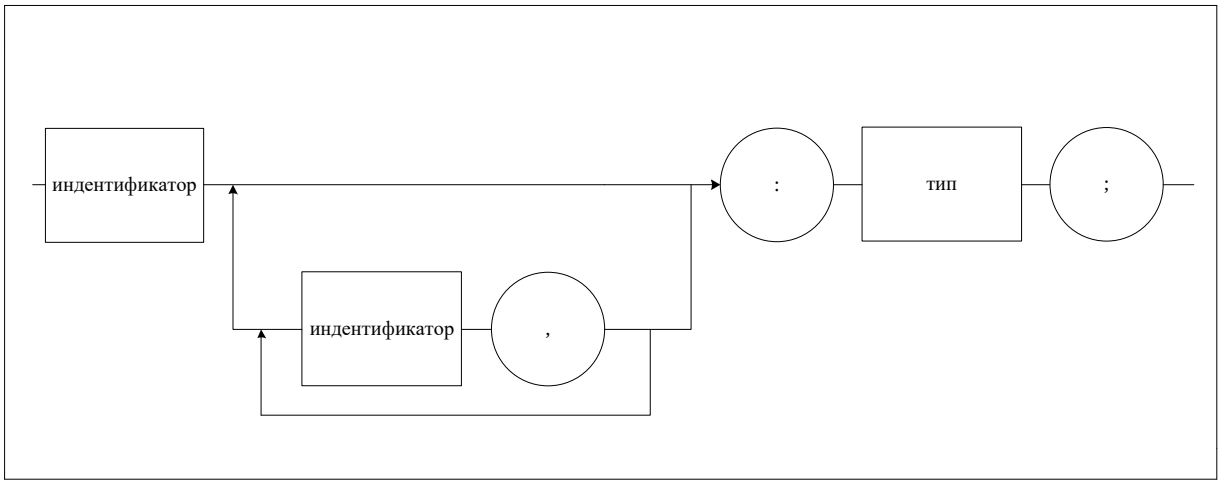


Рисунок 2 - Синтаксическая диаграмма <описание>

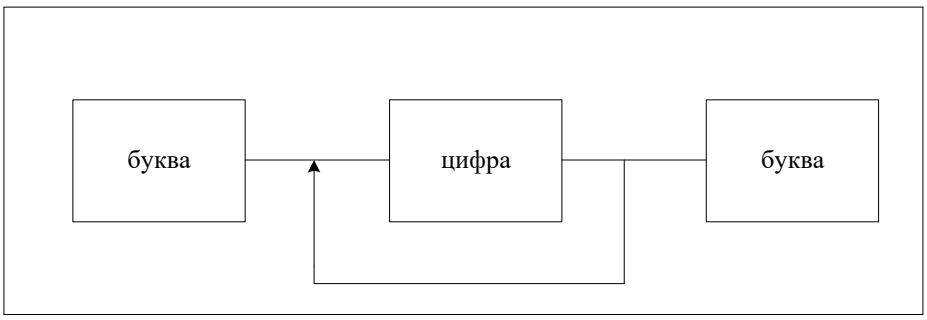


Рисунок 3 - Синтаксическая диаграмма <идентификатор>

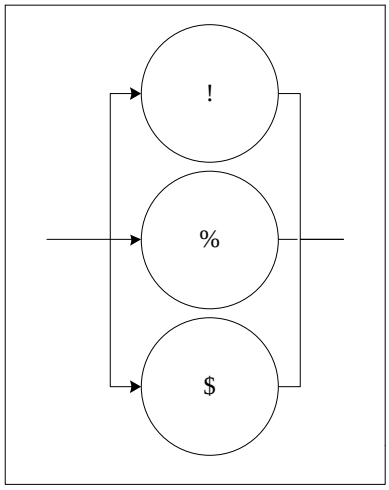


Рисунок 4 - Синтаксическая диаграмма <тип>

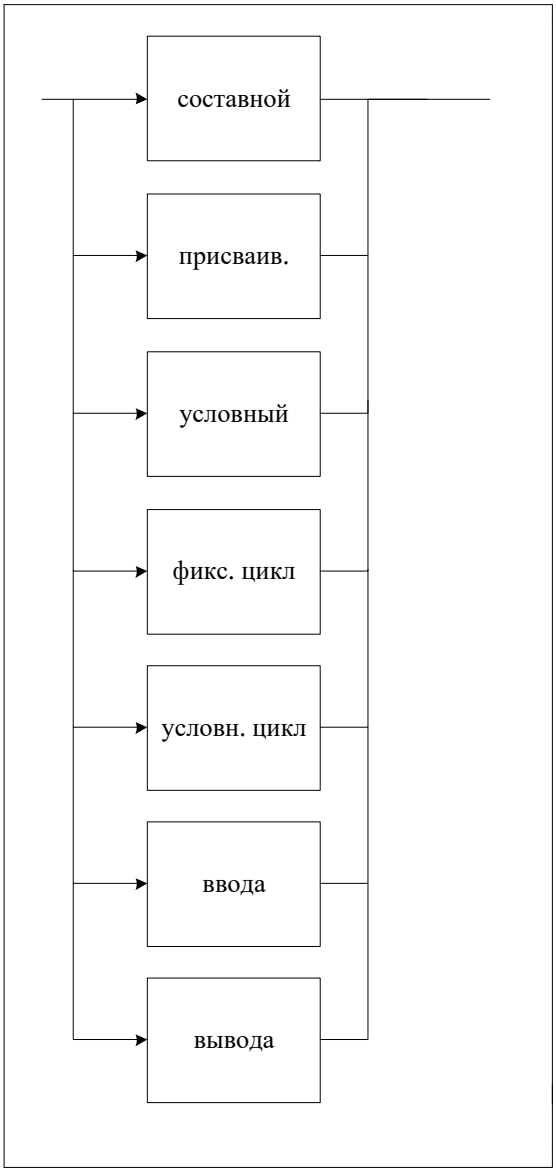


Рисунок 5 - Синтаксическая диаграмма <оператор>

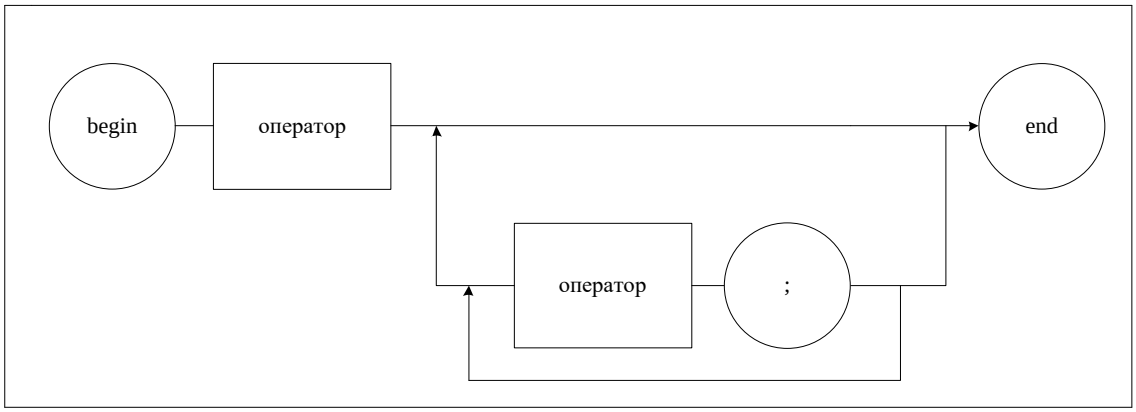


Рисунок 6 - Синтаксическая диаграмма <составной>

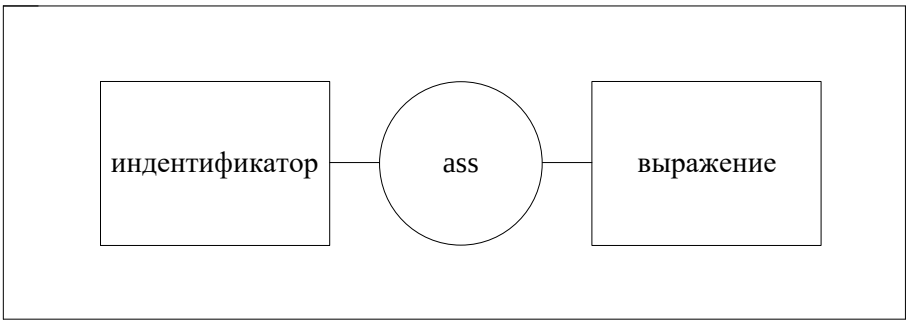


Рисунок 7 - Синтаксическая диаграмма <присваивание>

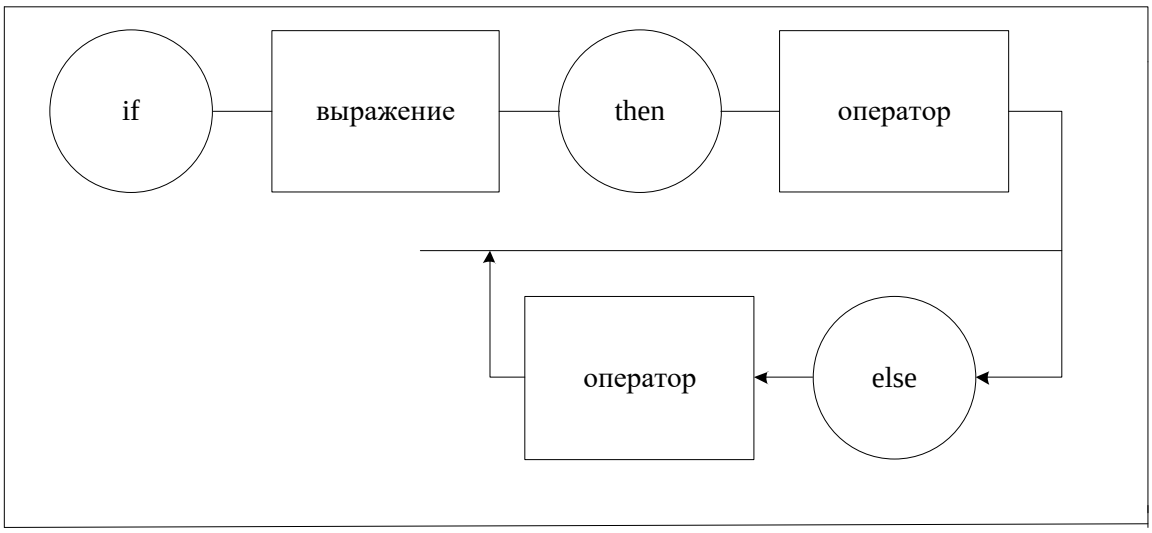


Рисунок 8 - Синтаксическая диаграмма <условный>

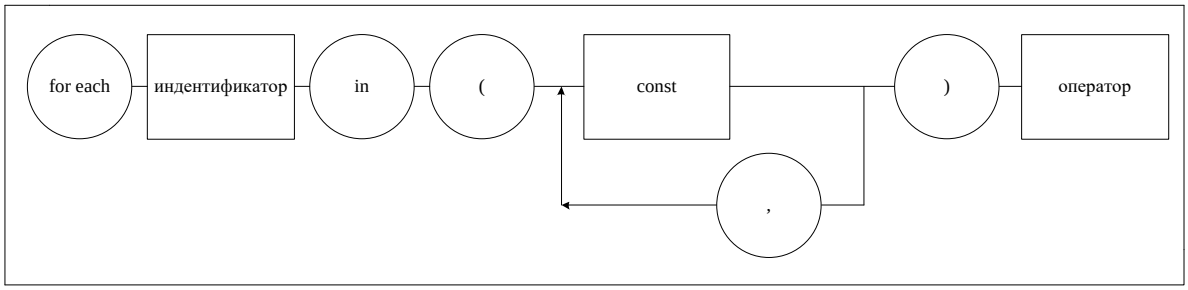


Рисунок 9 - Синтаксическая диаграмма <фиксированного_ цикла>

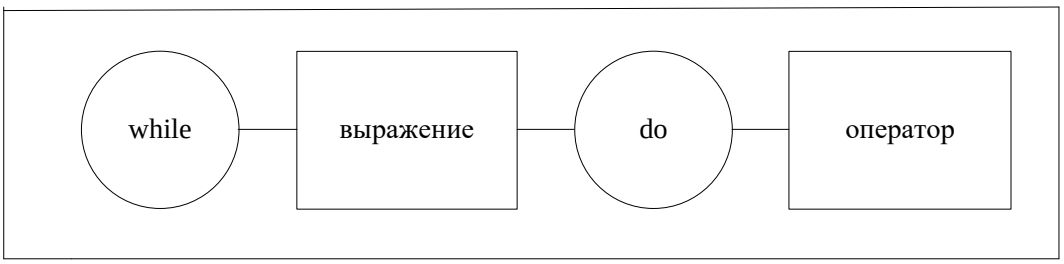


Рисунок 10 - Синтаксическая диаграмма <условного_ цикла>

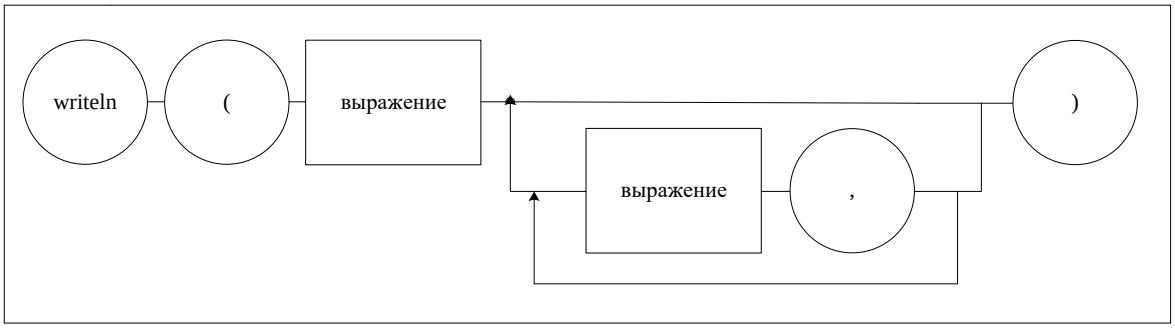


Рисунок 11 - Синтаксическая диаграмма <ввод>

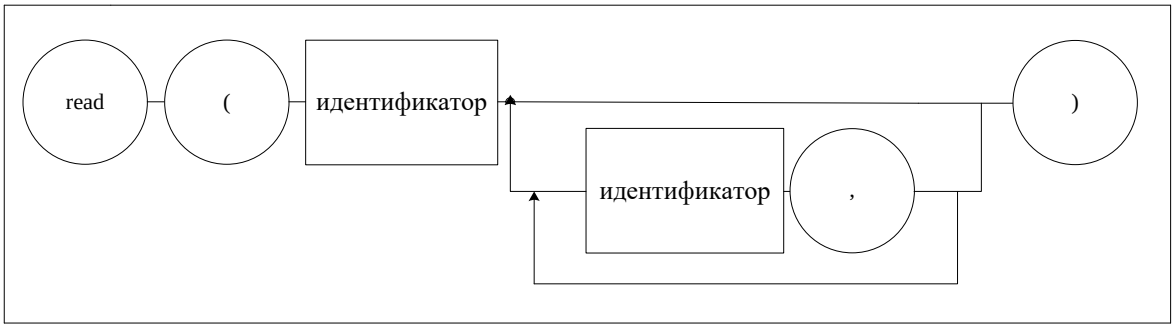


Рисунок 12 - Синтаксическая диаграмма <вывод>

# 2 Разработка лексического анализатора

## 2.1 Разработка лексического

Лексический анализатор (ЛА) – это первый этап процесса компиляции, на котором символы, составляющие исходную программу, группируются в отдельные минимальные единицы текста, несущие смысловую нагрузку – лексемы, с целью получения на выходе идентифицированных последовательностей, называемых токенами. Распознавание лексем в контексте грамматики обычно производится путём их классификации согласно идентификаторам токенов, определяемых грамматикой языка. Задача лексического анализа - выделить лексемы и преобразовать их к виду, удобному для последующей обработки. В процедурных языках лексемы обычно делятся на классы: ключевые слова, разделители, числа, идентификаторы.

Каждая лексема представляет собой пару чисел вида (n, k), где n – номер таблицы лексем, k - номер лексемы в таблице.

Таблица №1 - таблица ключевых слов.

Таблица №2 - таблица разделителей.

Таблица №3 - таблица идентификаторов.

Таблица №4 - таблица чисел.

Считывается очередной символ. Если данный символ – буква, это распознаётся как начало служебного слова или идентификатора, если цифра – начало числа, если первый символ разделителя – разделитель. Если распознаётся идентификатор:

До первого встреченного символа разделителя символы добавляются в буфер. Затем проверяется на совпадение с одним из служебных слов и в случае совпадения добавляется в список лексем в виде (1,k), где k – номер служебного слова в таблице. Иначе проверяется на соответствие правилу описания идентификаторов. В случае несоответствия выдаётся сообщение об ошибке,

иначе определяется положение в таблице идентификаторов по методу суммирования всех символов идентификатора (если номер в таблице занят, увеличить k на 5. если в таблице недостаточно места для k, где k – полученный номер, увеличить длину до k). Если распознаётся число:

Если в буфере хранился знак ‘+‘, изменить переменную знака на 1, иначе – на -1.

До первого встреченного символа разделителя символы добавляются в буфер. При добавлении происходит проверка. Если встречено не число, не первый символ разделителя, не пробел, не символ новой строки и не ‘.’, выдать сообщение об ошибке. Иначе до встречи ‘.’ или первого символа разделителя добавить символ в целую часть числа (добавление символа в число производится умножением числа на 10 и прибавлением кода символа -48). Если встречена ‘.’, добавление происходит в дробную часть числа (добавление символа в дробную часть числа осуществляется следующим образом: до завершения буфера добавить к числу код символа -48, умноженный на число p, где p – текущий разряд. Изначально p=-1, при добавлении символа к числу уменьшается на 1) Затем число умножается на знак. После число добавляется в таблицу и в список лексем в виде (3,k), где k – номер служебного слова в таблице. положение в таблице чисел определяется по методу суммирования всех символов идентификатора (если в таблице недостаточно места для k, где k – полученный номер, увеличить длину до k). Если распознаётся разделитель:

Если первый символ разделителя ‘+’, ‘-‘ и до этого было число или идентификатор, разделитель добавляется в список лексем в виде (2,k) где k – номер разделителя в таблице. Иначе если разделитель односимвольный и не ‘/’, он также добавляется в список лексем. Иначе если первый символ разделителя ‘<’,’>’ или ‘=’ и следующий символ не ‘=’, разделитель состоит из одного символа, иначе – из 2.

2.2 Функции, реализующие синтаксический анализ

2.3 Тестирование

Тестирование проверки на правильность написания идентификаторов представлено на рисунке 2.

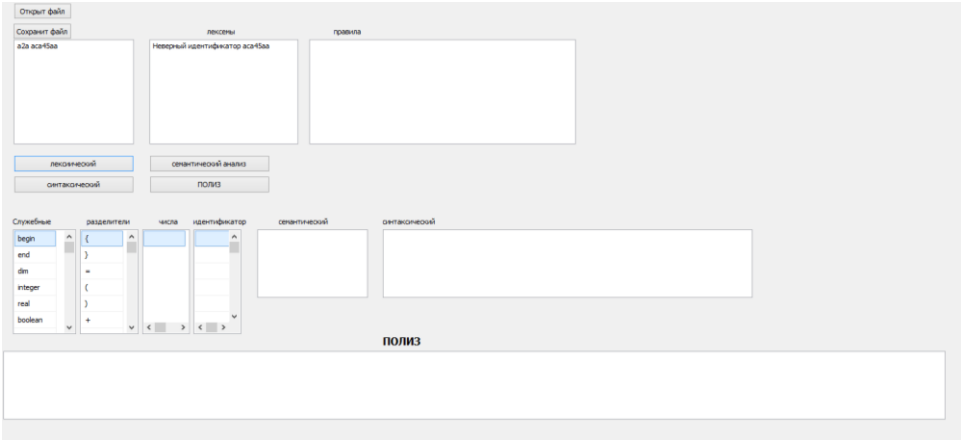


Рисунок 13 - Тестирование проверки на правильность написания идентификаторов

Тестирование проверки на правильность написания чисел представлено на рисунке 14-15.

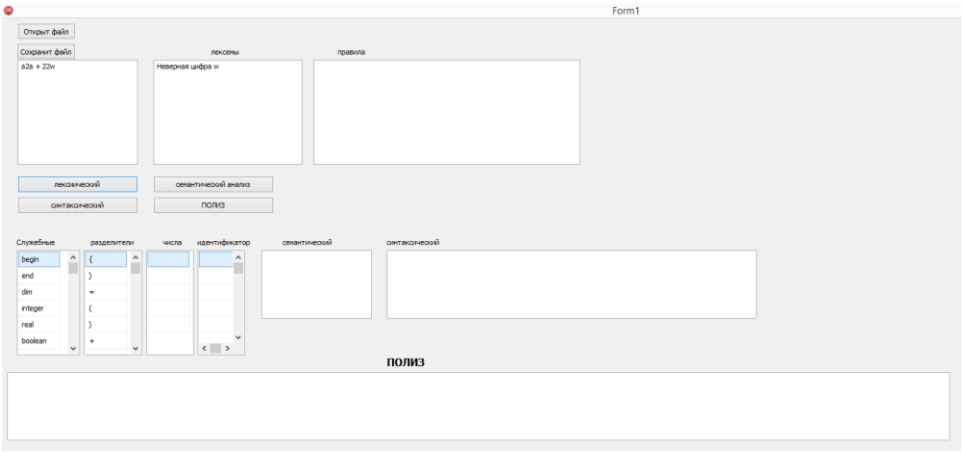


Рисунок 14 – Тестирование проверки на правильность написания чисел

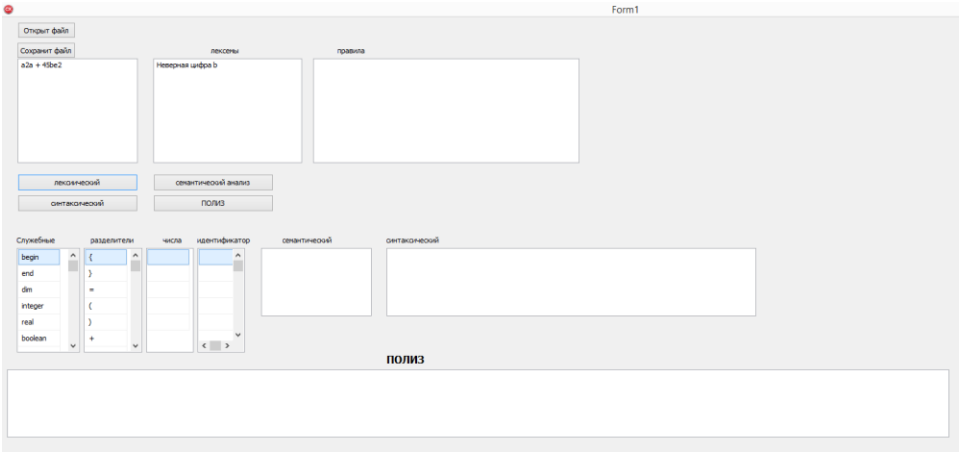


Рисунок 15 - Тестирование проверки на правильность написания чисел

Тестирование проверки на правильность написания разделителей представлено на рисунках 16-17.

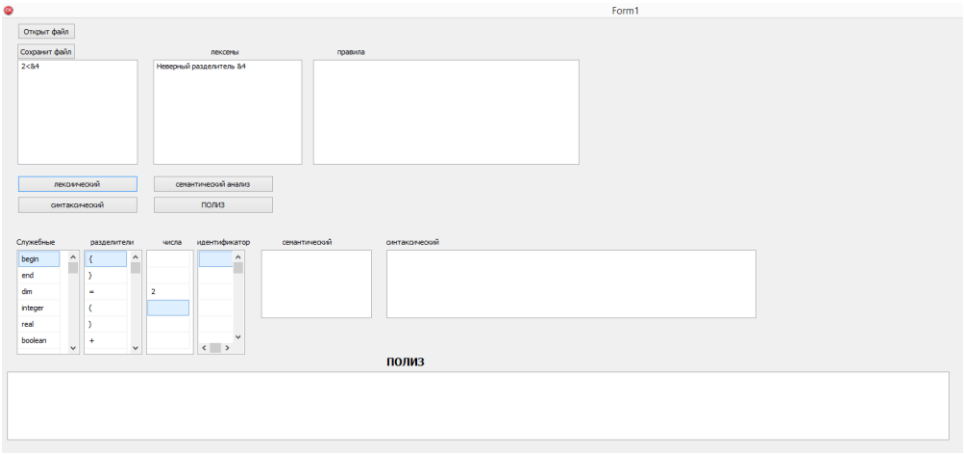


Рисунок 16 - Тестирование проверки на правильность написания разделителей

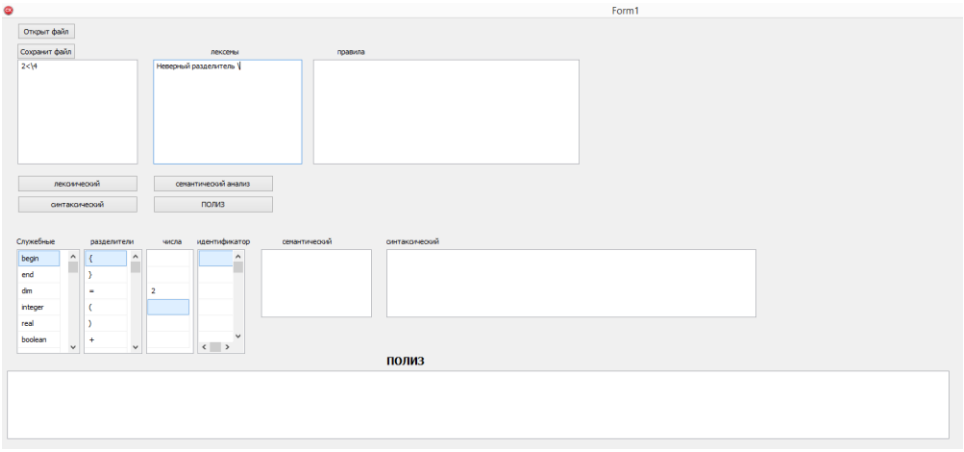


Рисунок 17 - Тестирование проверки на правильность написания разделителей

Тестирование проверки на завершённость комментария представлено на рисунке 18.

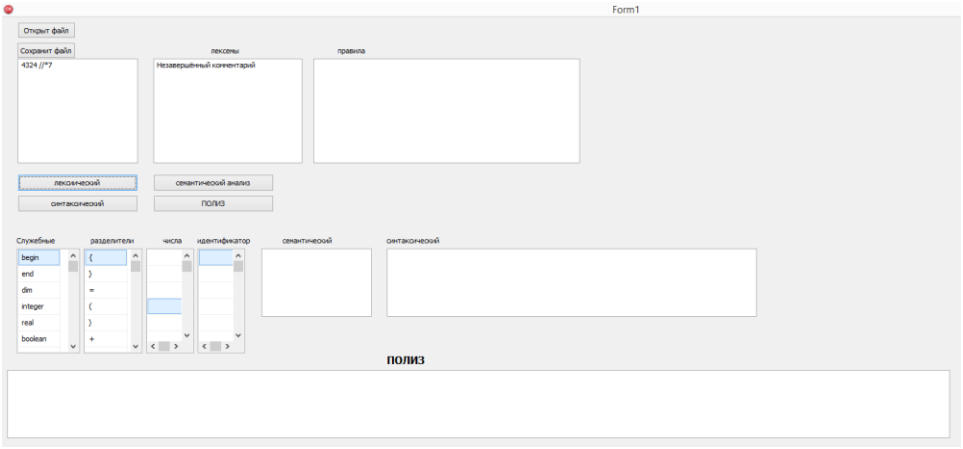


Рисунок 18 - Тестирование проверки на завершённость комментария

Следующим этапом компиляции является синтаксический анализ.

## 3. Разработка синтаксического анализатора

Синтаксический анализ – процесс сопоставления последовательности лексем языка с его формальной грамматикой. Программу, выполняющую данный процесс, называют синтаксическим анализатором. Существует несколько методов построения такой программы. В данной курсовой работе используется метод операторного предшествования.

Данный метод основан на том, что между любыми нетерминальными символами грамматики a и b устанавливается одно из 4 отношений:

1) a<b,если a – крайний левый символ некоторой основы(это отношение можно назвать “предшествует”);
2) a>b,если b – крайний правый символ некоторой основы(это отношение можно назвать “следует”);
3) a=b, если a и b принадлежат одной основе(это отношение между символами можно назвать “составляют основу”);
4) a0b (отсутствие предшествования), если эти символы не могут находиться рядом ни в одной основе.

Отношение предшествования некоммутативно.

Если между любыми двумя символами можно установить только одно из отношений предшествования, данный метод применим.

На основании этих отношений строится матрица предшествования, строки в которой соответствуют первым (левым) символам грамматики, столбцы – вторым (правым). На пересечении соответствующих строки и столбца помечается отношение.

Для построения матрицы используется следующий алгоритм, определяющий множества крайних левых и крайних правых и множества крайних левых и крайних правых терминальных символов для всех нетерминальных символов грамматики.

1 Для каждого нетерминального символа A ищутся все правила, содержащие A в левой части. Во множество L(A) включается самый левый символ правила, R(A) – самый правый.

2 Если L(A) содержит символы $A_1,A_2,\ldots,A_n$, L(A) необходимо дополнить символами, входящими во множества $L(A_1)$, $L(A_2),\ldots, L(A_n)$, и не входящими в L(A). То же самое выполнить для R(A).

3 Если хотя бы 1 множество изменилось, перейти к шагу 2, иначе – алгоритм завершён.

Для построения множества крайних левых и крайних правых терминальных символов используется следующий алгоритм:

1 Для каждого нетерминального символа A ищутся все правила, содержащие A в левой части. Во множество $L_t(A)$ включается самый левый символ правила, $R_t(A)$ – самый правый.

2 Множество $L_t(A)$ дополняется символами из множеств $L_t(A_1)$, $L_t(A_2),\ldots, L_t(A_n)$, где $A_1,A_2,\ldots,A_n$, - нетерминальные символы, входящие во множество L(A). То же самое выполнить для $R_t(A)$.

Множество крайних левых и крайних правых терминальных символов грамматики представлено на рисунке 8.

| | | |
|---|---|---|
| $P_1$ | { | } |
| $P_2$ | ;,dim,begin,switch ,for,while,read,writeln,a,n,( | ; |
| $P_3$ | dim,begin,switch ,for,while,read,writeln,a,n,( | integer,real,boolean,end,),do,writeln,as,,,a,n,z,} |
| $A_1$ | dim | integer,real,boolean |
| $B_1$ | begin,switch ,for,while,read,writeln,a,n,( | end, do,writeln, as, a,,, ),n,z,} |
| $B_2$ | begin,;,switch ,for,while,read,writeln,a,n,( | ;, end,do,writeln, as, a,,, ),n,z,} |
| $D_6$ | ,,a | a, , |
| $S_1$ | a,switch ,for,while,read,writeln,a,n,( | do,writeln, as, end, a,,, ),n,z,} |
| $C_3$ | , (,a,n | ),as,a,,,n,z |
| $F_1$ | ,,(,a,n | ,,),a,n,z |
| $F_2$ | (,a,n | ),a,n,z |
| $N_1$ | case | ; |
| $D_7$ | a | a |
| $C_1$ | a | as, ),a,n,z |
| $B_3$ | ;, begin,switch ,for,while,read,writeln,a,n,( | ;, end, do,writeln, as, a,,, ),n,z,} |

Рисунок 8 – множество крайних левых и крайних правых терминальных символов грамматики

Для заполнения матрицы предшествования используется следующий алгоритм:

1 Изначально между всеми терминальными символами устанавливается отношение “0”.

2 Взять произвольный терминальный символ. В дальнейшем считать его текущим.

2 Для текущего a ищутся правила, в которых символ a находится левее какого - либо терминального символа b и либо между ними нет других символов, либо находится один нетерминальный символ

3 Для каждого найденного на прошлом шаге символа b между символами a и b устанавливается отношение “=”. На пересечении строки матрицы, помеченной символом a и столбца, помеченного символом b, ставится знак “=”.

4 Для текущего a ищутся правила, в которых символ a находится левее какого - либо нетерминального символа A и между ними нет других символов.

5 Для каждого найденного на прошлом шаге символа A между символами a и всеми символами множества $L_t(A)$ устанавливается отношение “<”. На пересечении строки матрицы, помеченной символом a и столбцов, помеченного символами c, где $c \in L_t(A)$ ставится знак “<”.

6 Для текущего a ищутся правила, в которых символ a находится правее какого - либо нетерминального символа A и между ними нет других символов.

7 Для каждого найденного на прошлом шаге символа A между символами a и всеми символами множества $R_t(A)$ устанавливается отношение “>”. На пересечении строк матрицы, помеченных символами c, где $c \in R_t(A)$ и столбца, помеченного символом a ставится знак “<”.

8 Повторить шаги 2 – 7 для всех терминальных символов грамматики.

9 Вводятся два дополнительных символа: /1(начало строки) и /0 (конец строки).

10 между символом /1 и всеми символами множества $L_t(S)$(S – начальный символ грамматики) устанавливается отношение “<”. На пересечении строки матрицы, помеченной символом /1 и столбцов, помеченных символами c, где $c \in L_t(S)$ ставится знак “<”.

11 между символом /0 и всеми символами множества $R_t(S)$(S – начальный символ грамматики) устанавливается отношение “>”. На пересечении строк матрицы, помеченных символами c, где $c \in R_t(S)$ и столбца, помеченного символом /0,ставится знак “>”.

Матрица операторного предшествования для данной грамматики представлена на рисунке 9.

Матрица предшествования затем используется для синтаксического анализа. Перед синтаксическим анализом необходимо преобразовать исходную грамматику, заменив все нетерминальные символы одним нетерминальным. Алгоритм синтаксического анализа:

1 Дописать в конец строки символ /0, в вершину стека - /1.

2 Взять из входной строки текущий символ a, из стека – самый верхний терминальный символ b.

3 Если символ a - /0 и b - /1, алгоритм завершён, строка принадлежит грамматике. Иначе перейти к шагу 4

4 Найти в матрице предшествования клетку на пересечении строки, помеченной символом b и столбца, помеченного символом a.

5 Если в данной клетке находится символ “0”, строка не принадлежит грамматике. Необходимо выдать сообщение об ошибке.

6 Если в данной клетке находится символ “=” или “<”, необходимо выполнить сдвиг. Символ a помещается в стек, текущим становится следующий за ним. Перейти к шагу 2.

7 Если в данной клетке находится символ “>”, необходимо выполнить свёртку. Для этого в стеке выбирается самый верхний терминальный символ, а также все символы, связанные отношением “=”, а также все нетерминальные символы, находящиеся рядом с выбранными. Все выбранные символы удаляются из стека.

8 Сформировать из выбранных символов строку, которая будет правой частью правила. Все символы должны быть записаны в строку в том же порядке, в котором были помещены в стек.

9 Найти в грамматике правило с такой же правой частью. Если такого правила нет, строка не принадлежит грамматике, поэтому необходимо выдать сообщение об ошибке. Иначе поместить в стек нетерминальный символ из левой части и перейти к шагу 2.

### 2.2.2 Функции, реализующие синтаксический анализ

### 2.2.3 Тестирование

Тестирование на проверку выявления отсутствия предшествования представлено на рисунке 10.

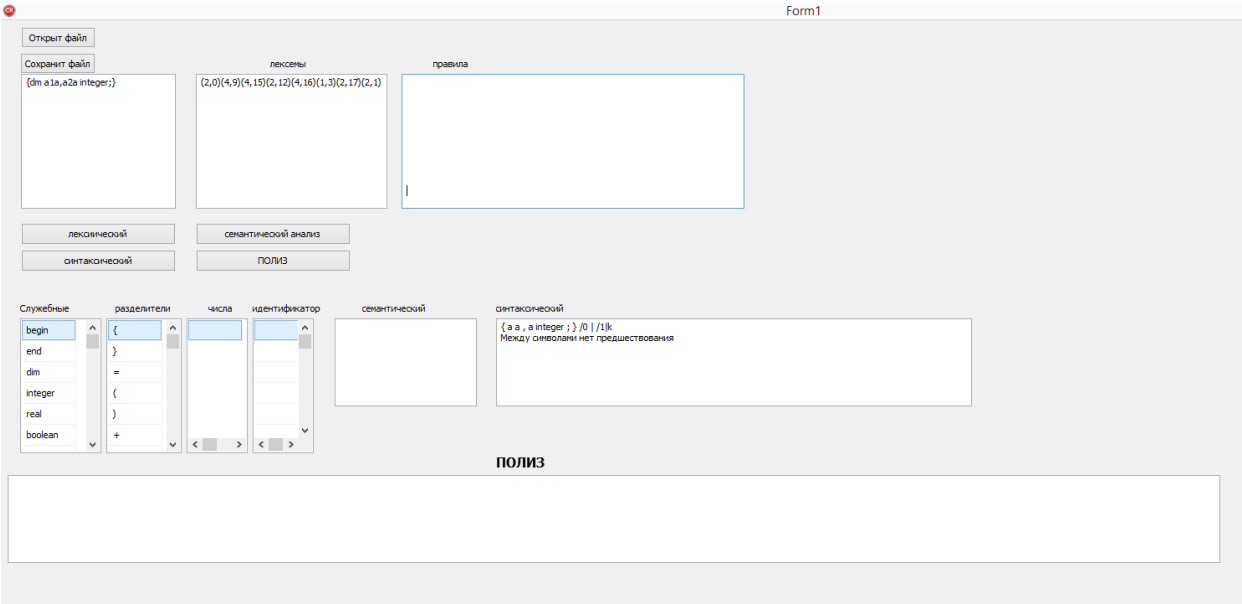


Рисунок 10 - тестирование на проверку выявления отсутствия предшествования

Тестирование на проверку выявления отсутствия правила представлено на рисунке 11.

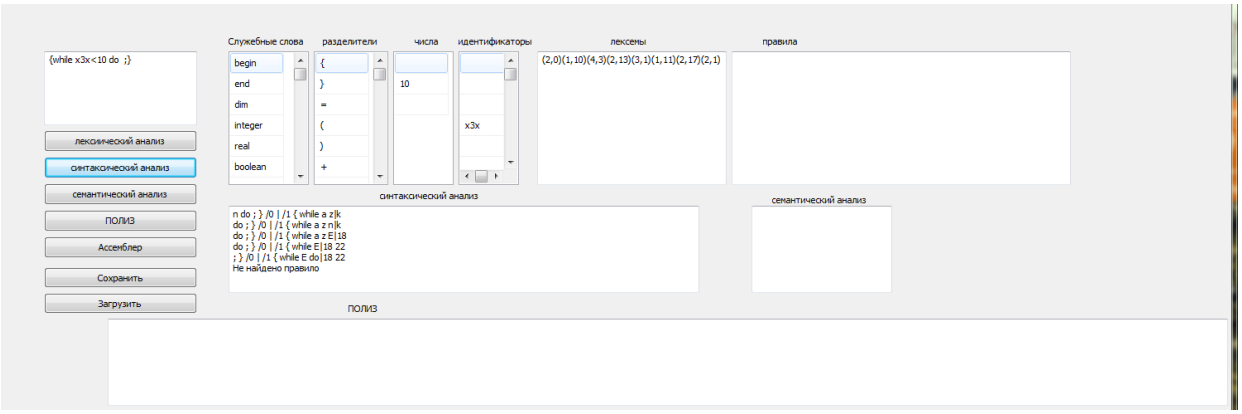


Рисунок 11 - Тестирование на проверку выявления отсутствия правила

Следующим этапом компиляции является семантический анализ.

## 3.1 Алгоритм синтаксического анализа

Синтаксический анализ – процесс сопоставления последовательности лексем языка с его формальной грамматикой. Программу, выполняющую данный процесс, называют синтаксическим анализатором. Существует несколько методов построения такой программы. В данной курсовой работе используется метод операторного предшествования.

Данный метод основан на том, что между любыми нетерминальными символами грамматики a и b устанавливается одно из 4 отношений:

1) a<b,если a – крайний левый символ некоторой основы(это отношение можно назвать “предшествует”);
2) a>b,если b – крайний правый символ некоторой основы(это отношение можно назвать “следует”);
3) a=b, если a и b принадлежат одной основе(это отношение между символами можно назвать “составляют основу”);
4) a0b (отсутствие предшествования), если эти символы не могут находиться рядом ни в одной основе.

Отношение предшествования некоммутативно.

Если между любыми двумя символами можно установить только одно из отношений предшествования, данный метод применим.

На основании этих отношений строится матрица предшествования, строки в которой соответствуют первым (левым) символам грамматики, столбцы – вторым (правым). На пересечении соответствующих строки и столбца помечается отношение.

Для построения матрицы используется следующий алгоритм, определяющий множества крайних левых и крайних правых и множества крайних левых и крайних правых терминальных символов для всех нетерминальных символов грамматики.

1 Для каждого нетерминального символа A ищутся все правила, содержащие A в левой части. Во множество L(A) включается самый левый символ правила, R(A) – самый правый.

2 Если L(A) содержит символы $A_1,A_2,\ldots,A_n$, L(A) необходимо дополнить символами, входящими во множества $L(A_1)$, $L(A_2)$,…, $L(A_n)$, и не входящими в L(A). То же самое выполнить для R(A).

3 Если хотя бы 1 множество изменилось, перейти к шагу 2, иначе – алгоритм завершён.

Для построения множества крайних левых и крайних правых терминальных символов используется следующий алгоритм:

1 Для каждого нетерминального символа A ищутся все правила, содержащие A в левой части. Во множество $L_t(A)$ включается самый левый символ правила, $R_t(A)$ – самый правый.

2 Множество $L_t(A)$ дополняется символами из множеств $L_t(A_1)$, $L_t(A_2)$,…, $L_t(A_n)$, где $A_1,A_2,\ldots,A_n$, - нетерминальные символы, входящие во множество L(A). То же самое выполнить для $R_t(A)$.

Множество крайних левых и крайних правых терминальных символов грамматики представлено на рисунке 8.

| | | |
|---|---|---|
| P1 | { | } |
| P2 | ;,dim,begin,switch ,for,while,read,writeln,a,n,( | ; |
| P3 | dim,begin,switch ,for,while,read,writeln,a,n,( | integer,real,boolean,end,),do,writeln,as,,,a,n,z,} |
| A1 | dim | integer,real,boolean |
| B1 | begin,switch ,for,while,read,writeln,a,n,( | end, do,writeln, as, a,,, ),n,z,} |
| B2 | begin,;,switch ,for,while,read,writeln,a,n,( | ;, end,do,writeln, as, a,,, ),n,z,} |
| D6 | ,,a | a, , |
| S1 | a,switch ,for,while,read,writeln,a,n,( | do,writeln, as, end, a,,, ),n,z,} |
| C3 | , (,a,n | ),as,a,,,n,z |
| F1 | ,,(,a,n | ,,),a,n,z |
| F2 | (,a,n | ),a,n,z |
| N1 | case | ; |
| D7 | a | a |
| C1 | a | as, ),a,n,z |
| B3 | ;, begin,switch ,for,while,read,writeln,a,n,( | ;, end, do,writeln, as, a,,, ),n,z,} |

Рисунок 19 – множество крайних левых и крайних правых терминальных символов грамматики

Для заполнения матрицы предшествования используется следующий алгоритм:

1 Изначально между всеми терминальными символами устанавливается отношение “0”.

2 Взять произвольный терминальный символ. В дальнейшем считать его текущим.

2 Для текущего a ищутся правила, в которых символ a находится левее какого - либо терминального символа b и либо между ними нет других символов, либо находится один нетерминальный символ

3 Для каждого найденного на прошлом шаге символа b между символами a и b устанавливается отношение “=”. На пересечении строки матрицы, помеченной символом a и столбца, помеченного символом b, ставится знак “=”.

4 Для текущего a ищутся правила, в которых символ a находится левее какого - либо нетерминального символа A и между ними нет других символов.

5 Для каждого найденного на прошлом шаге символа A между символами a и всеми символами множества $L_t(A)$ устанавливается отношение “<”. На пересечении строки матрицы, помеченной символом a и столбцов, помеченного символами c, где $c \in L_t(A)$ ставится знак “<”.

6 Для текущего a ищутся правила, в которых символ a находится правее какого - либо нетерминального символа A и между ними нет других символов.

7 Для каждого найденного на прошлом шаге символа A между символами a и всеми символами множества $R_t(A)$ устанавливается отношение “>”. На пересечении строк матрицы, помеченных символами c, где $c \in R_t(A)$ и столбца, помеченного символом a ставится знак “<”.

8 Повторить шаги 2 – 7 для всех терминальных символов грамматики.

9 Вводятся два дополнительных символа: /1(начало строки) и /0 (конец строки).

10 между символом /1 и всеми символами множества $L_t(S)$(S – начальный символ грамматики) устанавливается отношение “<”. На пересечении строки

матрицы, помеченной символом /1 и столбцов, помеченных символами c, где $c \in L_t(S)$ ставится знак “<”.

11 между символом /0 и всеми символами множества $R_t(S)$(S – начальный символ грамматики) устанавливается отношение “>”. На пересечении строк матрицы, помеченных символами c, где $c \in R_t(S)$ и столбца, помеченного символом /0,ставится знак “>”.

Матрица операторного предшествования для данной грамматики представлена на рисунке 9.

Матрица предшествования затем используется для синтаксического анализа. Перед синтаксическим анализом необходимо преобразовать исходную грамматику, заменив все нетерминальные символы одним нетерминальным.

Алгоритм синтаксического анализа:

1 Дописать в конец строки символ /0, в вершину стека - /1.

2 Взять из входной строки текущий символ a, из стека – самый верхний терминальный символ b.

3 Если символ a - /0 и b - /1, алгоритм завершён, строка принадлежит грамматике. Иначе перейти к шагу 4

4 Найти в матрице предшествования клетку на пересечении строки, помеченной символом b и столбца, помеченного символом a.

5 Если в данной клетке находится символ “0”, строка не принадлежит грамматике. Необходимо выдать сообщение об ошибке.

6 Если в данной клетке находится символ “=” или “<”, необходимо выполнить сдвиг. Символ a помещается в стек, текущим становится следующий за ним. Перейти к шагу 2.

7 Если в данной клетке находится символ “>”, необходимо выполнить свёртку. Для этого в стеке выбирается самый верхний терминальный символ, а также все символы, связанные отношением “=”, а также все нетерминальные

символы, находящиеся рядом с выбранными. Все выбранные символы удаляются из стека.

8 Сформировать из выбранных символов строку, которая будет правой частью правила. Все символы должны быть записаны в строку в том же порядке, в котором были помещены в стек.

9 Найти в грамматике правило с такой же правой частью. Если такого правила нет, строка не принадлежит грамматике, поэтому необходимо выдать сообщение об ошибке. Иначе поместить в стек нетерминальный символ из левой части и перейти к шагу 2.

Семантический анализ – этап компиляции, на котором проверяются правила языка, не входящие в формальную грамматику. В данной курсовой работе проверялись 2 правила:

1 Проверка на переопределение переменной.

2 Проверка на использование необъявленной переменной.

Для проверки этих правил используется стек.

Алгоритм:

1 Если символ начала обьявления найден, проверьте первое правило до его завершения, в противном случае проверьте второе. В обоих этих случаях выполняется поиск найденного идентификатора в стеке.

2 Если при просмотре обьявления обнаружено наличие идентификатора в стеке, значит, происходит попытка переопределить переменную. Выдать сообщение об ошибке. Иначе идентификатор добавляется в стек.

3 Если при просмотре других частей программы обнаружено отсутствие идентификатора в стеке, значит, происходит использование необъявленной переменной. Выдать сообщение об ошибке.

4 Если просмотрен весь текст программы и не было ошибок, программа соответствует правилам и алгоритм завершён. Иначе перейти к шагу 1.

3.2 Функции, реализующие синтаксический анализ

3.3 Тестирование

Тестирование на проверку переопределения переменной представлено на рисунке 20.

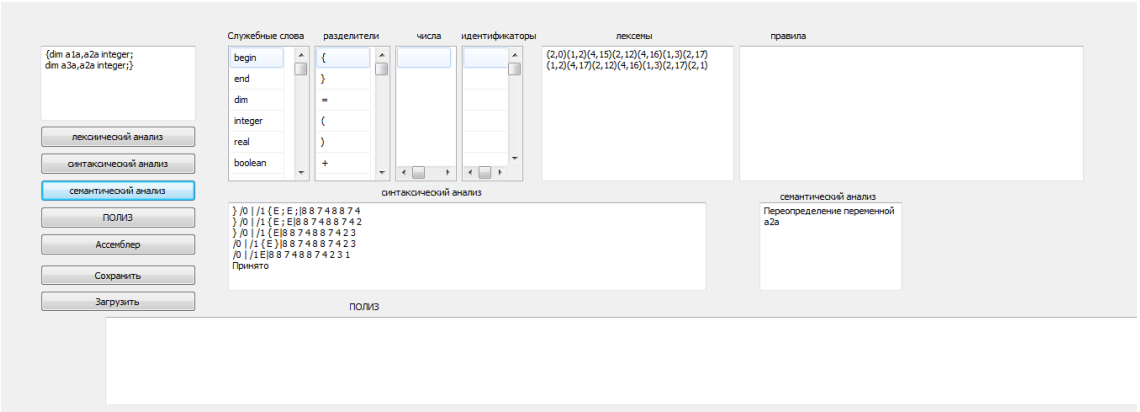


Рисунок 20 - Тестирование на проверку переопределения переменной.

Тестирование на проверку использования необъявленного символа представлено на рисунке 21.

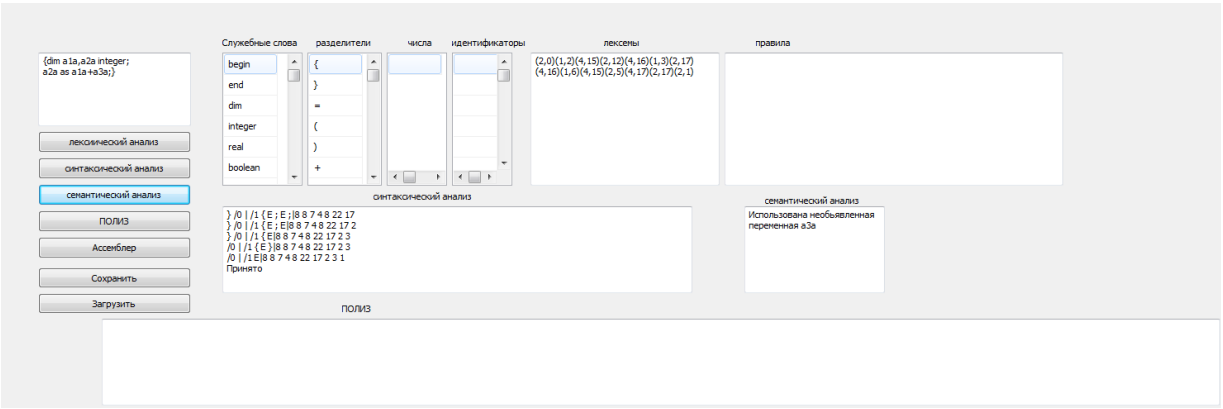


Рисунок 21 - Тестирование на проверку использования необъявленной переменной.

Следующим этапом компиляции является семантический анализ.

# 4. Семантический анализ

## 4.1 Алгоритм семантического анализа

Семантический анализ – этап компиляции, на котором проверяются правила языка, не входящие в формальную грамматику. В данной курсовой работе проверялись 2 правила:

1 Проверка на переопределение переменной.

2 Проверка на использование необъявленной переменной.

Для проверки этих правил используется стек.

Алгоритм:

1 Если символ начала обьявления найден, проверьте первое правило до его завершения, в противном случае проверьте второе. В обоих этих случаях выполняется поиск найденного идентификатора в стеке.

2 Если при просмотре обьявления обнаружено наличие идентификатора в стеке, значит, происходит попытка переопределить переменную. Выдать сообщение об ошибке. Иначе идентификатор добавляется в стек.

3 Если при просмотре других частей программы обнаружено отсутствие идентификатора в стеке, значит, происходит использование необъявленной переменной. Выдать сообщение об ошибке.

4 Если просмотрен весь текст программы и не было ошибок, программа соответствует правилам и алгоритм завершён. Иначе перейти к шагу 1.

## 4.2 Функции, реализующие синтаксический анализ

## 4.3 Тестирование

Тестирование на проверку переопределения переменной представлено на рисунке 12.

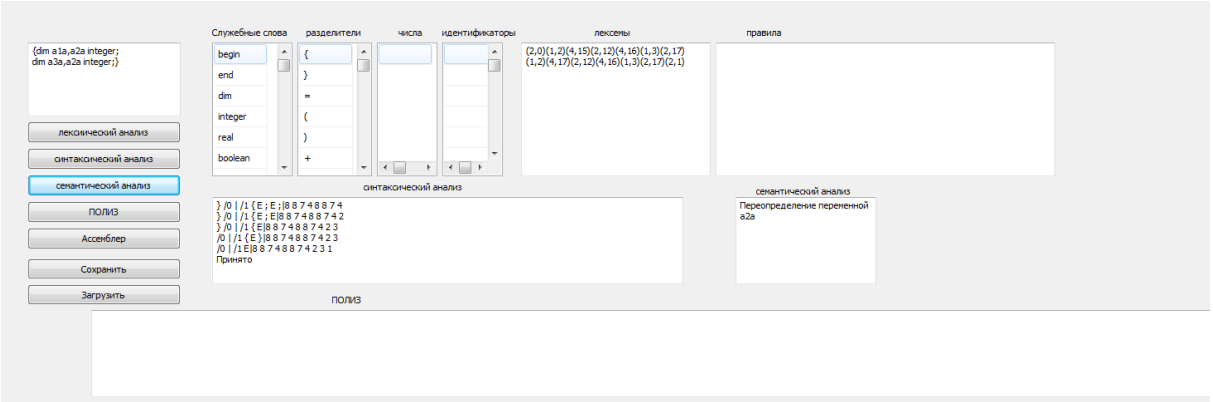


Рисунок 12 - Тестирование на проверку переопределения переменной.

Тестирование на проверку использования необьявленного символа представлено на рисунке 13.

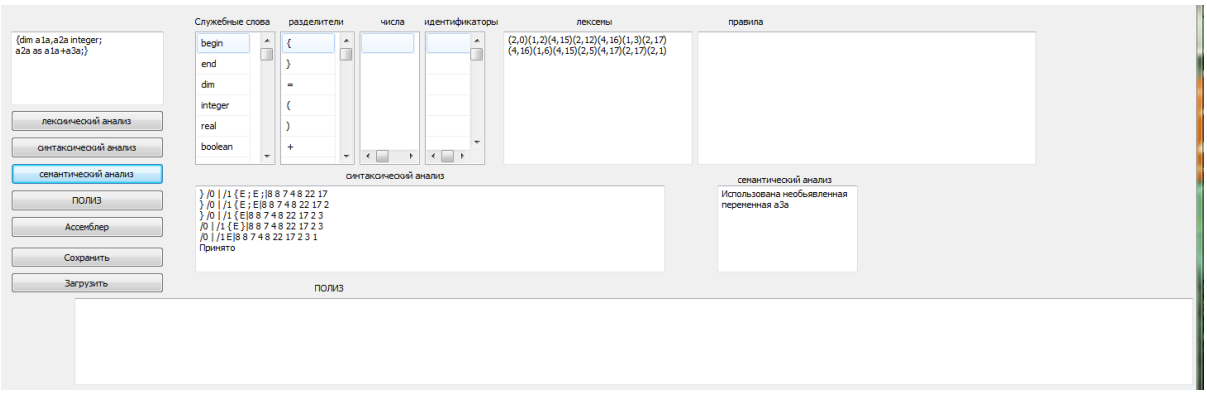


Рисунок 12 - Тестирование на проверку использования необъявленной переменной.

Следующим этапом компиляции является Перевод в ПОЛИЗ.

## 5. Перевод в ПОЛИЗ

ПОЛИЗ (Обратная польская запись) — форма записи математических выражений, в которой операнды расположены перед знаками операций.

В компиляторах ПОЛИЗ используется как промежуточное представление программы перед генерацией объектного кода.

Отличительной особенностью ПОЛИЗ является то, что операнды расположены перед знаком операции. Выражение, записанное в

ПОЛИЗ, выполняется слева направо, благодаря чему нет необходимости в использовании скобок.

Бинарный оператор AoB, где A и B – операнды, o – знак операции записывается так:

ABo.

Унарный оператор oA, где A – операнд, o – знак операции записывается так:

oA.

Так как в ПОЛИЗе нельзя отличить унарные и бинарные операции, обозначаемые одним символом, для унарных операций необходимо использовать другие знаки (в данной работе для унарной операции "-" используется операция "@").

Для перевода в ПОЛИЗ операторов используются следующие правила:

1) Объявление переменных dim $A_1,A_2,\ldots A_n$ B, где $A_1,A_2,A_n$ – переменные, B – тип, записывается в ПОЛИЗе как $A_1$ B, $A_2$ B,…$A_n$ B.
2) оператор присваивания A=B, где A – идентификатор, B – выражение, записывается в ПОЛИЗе как A&$B_1$=,где & - операция взятия адреса переменной A, $B_1$ – записанное в ПОЛИЗе выражение B, = - операции записи по адресу переменной A значения $B_1$
3) Оператор ввода read($A_1,A_2,\ldots A_n$), где $A_1,A_2,\ldots A_n$ – выражения, записывается в ПОЛИЗе как $B_1$& read, $B_2$& read,… $B_n$& read ,где & - операция взятия адреса ?, $B_1,B_2,B_n$ – записанные в ПОЛИЗе выражения $A_1,A_2,\ldots A_n$.
4) Оператор вывода writeln $A_1,A_2,\ldots A_n$, где $A_1,A_2,\ldots A_n$ – выражения, записывается в ПОЛИЗе как $B_1$ writeln, $B_2$ writeln,… $B_n$ writeln,где $B_1,B_2,B_n$ – записанные в ПОЛИЗе выражения $A_1,A_2,\ldots A_n$

Для перевода в ПОЛИЗ цикла for, while и оператора выбора switch требуется введение меток и операторов перехода.

Метка является символьным обозначением адреса памяти, в котором находится помеченная программа. В данной работе все метки имеют вид mi, где i – целое положительное число.

Операторы перехода делятся на безусловные (в данной работе goto) и условные (в данной работе goto1).

Оператор goto с операндом m – оператор, смысл которого в том, что он изменяет ход выполнения программы, осуществляя переход на метку m независимо от каких – либо условий.

Оператор goto с операндом m – оператор, смысл которого в том, что он изменяет ход выполнения программы, осуществляя переход на метку m независимо от каких – либо условий.

Оператор goto1 с операндом n,m отличается от goto тем, что переход на метку m осуществляется только при истинности условия n.

Оператор while (A) do B в ПОЛИЗе выглядит следующим образом:

$m_k$:$A_1$! $m_{k+1}$ goto1 B $m_k$ goto $m_{k+1}$:,

где $A_1$ – переведённое в ПОЛИЗ выражение A, B – оператор, выполняющийся в случае истинности условия цикла, ! – отрицание.

Оператор for a=(A,B,C) D в ПОЛИЗе выглядит следующим образом:

a&$A_1$ = $m_k$:a $B_1$ <= ! $m_{k+1}$ goto1 D a&a$C_1$+= $m_k$ goto $m_{k+1}$:

a – идентификатор

где $A_1$ – переведённое в ПОЛИЗ выражение A (первое значение переменной), $B_1$ – переведённое в ПОЛИЗ выражение B (последнее значение переменной), D – оператор, выполняющийся в случае истинности условия цикла, ! – отрицание, $C_1$ – переведённое в ПОЛИЗ выражение C (шаг). Если C отсутствует, ПОЛИЗ будет иметь вид

a&$A_1$ = $m_k$:a $B_1$ <= ! $m_{k+1}$ goto1 D a&a 1+= $m_k$ goto $m_{k+1}$:

Оператор switcn a=(A,B,C) D в ПОЛИЗе выглядит следующим образом:

a&$A_1$ = $m_k$:a $B_1$ <= ! $m_{k+1}$ goto1 D a&a$C_1$+= $m_k$ goto $m_{k+1}$:

a – идентификатор

где $A_1$ – переведённое в ПОЛИЗ выражение A (первое значение переменной), $B_1$ – переведённое в ПОЛИЗ выражение B (последнее значение переменной), D – оператор, выполняющийся в случае истинности условия цикла, ! – отрицание, $C_1$ – переведённое в ПОЛИЗ выражение C (шаг). Если C отсутствует, ПОЛИЗ будет иметь вид

a&$A_1$ = $m_k$:a $B_1$ <= ! $m_{k+1}$ goto1 D a&a 1+= $m_k$ goto $m_{k+1}$:

Оператор выбора switch A{case $n_1$:B;case $n_2$:B…, case $n_k$:B} в ПОЛИЗ будет иметь вид:

$m_k$:$A_1$ $n_1$== ! $m_{l+1}$ goto1 B goto $m_n$ $m_{l+2}$:$A_1$ $n_2$== ! $m_{k+2}$ goto1 B goto $m_n$ …

…$A_1$ $n_k$== ! $m_{l+k}$ goto1 B $m_n$:$m_{l+k}$.

1. > identifier(0) для хранения идентификаторов;
2. Таблица table <AnsiString> limiters(0) для хранения разделителей;
3. Таблица table <AnsiString> keywords(0) для хранения ключевых слов;
4. Структура конечных автоматов enum state{};
5. Функция int letter(), которая проверяет, является ли текущий символ буквой. Входными данными является текущий символ. Выходными данными является булевая переменная.
6. Функция int numeral(), которая проверяет, является ли текущий символ цифрой. Входными данными является текущий символ. Выходными данными является булевая переменная.
7. Функция read_c() которая позволяет считывать следующий элемент входных данных.
8. Функция void reduce_ cur_var(), которая уменьшает номер считываемого символа. Входными данными является текущий символ.
9. Функция void add_table(int num_table), которая добавляет элемент в заданную таблицу. Входными данными является номер таблицы.

10. Используется структура таблицы лексем struct Table_Lecsem {int num_table; int num_lecsem;}
11. Функция void out_lecsem(int num_table,int num_lecsem) ,которая выводит лексемы на экран. Входными данными является номер таблицы и номер лексемы.
12. Основная функция лексического анализатора void Lecs_Analizator(). Позволяет распределять лексемы по нужным таблицам. Если введен символ, не принадлежащий ни одной таблице, то сообщение об ошибке выводится на экран.

# ЗАКЛЮЧЕНИЕ

В процессе разработки программного обеспечения, для решения конкретной задачи была изучена следующая литература по теме курсовой работы: теория по лексическим, синтаксическим и семантическим анализаторам, документация по среде разработки. В ходе написания курсовой работы, были выполнены следующие задачи:

- разработка лексического анализатора, в процессе которого весь текст исходной программы преобразуется в список лексем;
- разработка синтаксического анализатора, который реализуется методом рекурсивного спуска;
- разработка алгоритма семантического анализатора, который проверяет правильность записи исходной программы;
- реализован перевод программы в польско-инверсную запись, которая происходит параллельно с синтаксическим и семантическим анализаторами;
- создание внешнего вида графического интерфейса;
- проведение анализа литературы по теме исследования.

Программа может быть усовершенствована в следующих направлениях:

– перевод кода данного модельного языка в ассемблер;

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Ахо А., Сети Р., Ульман Д. Компиляторы: принципы, технологии и инструменты.: Пер. с англ. – М.: Изд. дом «Вильямс», 2001.
2. Cyberforum.ru – форум программистов и сисадминов [Сайт]. URL: http://cyberforum.ru (дата обращения 17.12.2019)
3. Е.Н.Ишакова, Разработка компиляторов. Методические указания к курсовой работе. - Оренбург: ГОУ ОГУ, 2005 - 50с.
4. Academic.ru - Академик. Большой Энциклопедический словарь. [Сайт]. URL: https://dic.academic.ru/dic.nsf/fin_enc/28322 (дата обращения 19.12.2019)
5. Теория языков программирования и методы трансляции [Сайт]. URL: http://ermak.cs.nstu.ru/trans/ (дата обращения 19.12.2019)

# ПРИЛОЖЕНИЕ А. ТЕКСТ ПРОГРАММЫ

```
//---------------------------------------------------------------
------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
//---------------------------------------------------------------
------------
#pragma package(smart_init)
#pragma resource "*.dfm"
#include <fstream>
TForm1 *Form1;
using namespace std;
AnsiString d[36];
enum
leks{beg,beg1,ident,number,number2,sep,sep1,sep2,er,en,ign,ign1,i
gn2};
AnsiString u;
class  leksems
{public:
AnsiString a;
bool id;
bool num;
public:
leksems*next;
void add(leksems**,AnsiString,bool,bool);
leksems()
{a="";
next=0;
id=0;
num=0;
}
leksems(AnsiString k,bool t,bool t1)
{a=k;
next=0;
id=t;
```

```
num=t1;
}
void print(leksems*,TMemo*);
void print1(leksems*,TMemo*);
void del(leksems**);
AnsiString forms(leksems*);
bool find(leksems*,AnsiString);
};
leksems*keywords;
leksems*dels;
leksems*b;
class  leksan
{ public:
int a;
int b;
public:
leksan*next;
void add(leksan**,int,int);
leksan()
{a=0;
b=0;
next=0;
}
leksan(int x,int y)
{a=x;
b=y;
next=0;
}
void print(leksan*,TMemo*);
};
 void leksems::add(leksems**a,AnsiString b,bool t,bool t1)
 {
 if ((*a)==0) {
 (*a)=new leksems(b,t,t1);
 return;
 }
 leksems*c=(*a);
 while(c->next!=0)
```

```
 c=c->next;
 leksems*d=new leksems(b,t,t1);
 c->next=d;
 }
 bool leksems::find(leksems*a,AnsiString b)
 {
 bool z=0;
 leksems*c=a;
 while(c!=0)
 {    if(c->a==b)
 z=1;
  c=c->next;
 }
 return z;
 }
   void leksems::del(leksems**a)
{
  leksems*b=(*a);
  (*a)=(*a)->next;
   delete b;
}
AnsiString leksems::forms(leksems*a)
{
AnsiString f="";
while(a!=0)
{
  f+=a->a;
  f+=" ";
  a=a->next;
}
  return f;
}
 void leksems::print(leksems*x,TMemo*Memo1)
 { AnsiString h="";
 while(x!=0)
{
h+=x->a;
Memo1->Text+=h;
```

```
h="";
x=x->next;}
 }
 void leksems::print1(leksems*x,TMemo*Memo1)
 { AnsiString h="";
 while(x!=0)
{
h+=x->a;
Memo1->Text+=h;
Memo1->Text+=" ";
h="";
x=x->next;}
 }
 void leksan::add(leksan**a,int x,int y)
 {
 if ((*a)==0) {
 (*a)=new leksan(x,y);
  return;
 }
 leksan*c=(*a);
 while(c->next!=0)
c=c->next;
 leksan*d=new leksan(x,y);
c->next=d;
 }
 void leksan::print(leksan*x,TMemo*Memo1)
 { AnsiString h="";
 while(x!=0)
{h+="(";
h+=x->a;
h+=",";
h+=x->b;
h+=")";
Memo1->Text+=h;
h="";
x=x->next;}
 }
  leksan*f;
```

```
  int analyzeres(AnsiString s,leksems*k)
  {
  int z=-1;
 int i=0;
 while(k!=0&&z==-1)
{
if(s==k->a)
{z=i;
f->add(&f,1,i);}
k=k->next;
i++;
}
return z;
  }
  int idents(AnsiString s,TStringGrid*StringGrid4,TMemo*Memo2)
  {
  int z=0;
  AnsiString g="Неверный идентификатор ";
  if(s.Length()<2)
  {g+=s;
  Memo2->Text=g;
  return -1;
  }
if(s[1]<'A'||(s[1]>'Z'&&s[1]<'a')||s[1]>'z')
 {
   g+=s;
    Memo2->Text=g;
   return -1;
 }
  else
 z+=s[1]%10;

if(s[s.Length()]<'A'||(s[s.Length()]>'Z'&&s[s.Length()]<'a')||s[s
.Length()]>'z')
 {
   g+=s;
    Memo2->Text=g;
   return -1;
```

```
 }
 else z+=s[s.Length()]%10;
   for(int i=2;i<s.Length();i++)
  { if(s[i]<'0'||s[i]>'9')
   {   g+=s;
   Memo2->Text=g;
     return -1;
   }
   else  z+=s[i]-48;
  }
  int c=5;
  int i=0;
  while(StringGrid4->Cells[0][z+c*i]!=""&&z+c*i<=StringGrid4-
>RowCount-1&&StringGrid4->Cells[0][z+c*i]!=s)
  i++;
  if(z+c*i>StringGrid4->RowCount-1)
  StringGrid4->RowCount=z+c*i+1;
  StringGrid4->Cells[0][z+c*i]=s;
  return (z+c*i);
  }
  int find(char f,TStringGrid*StringGrid2,int y)
{int x=-1;
for(int i=0;i<StringGrid2->RowCount;i++)
{
 if      ((StringGrid2->Cells[0][i]).Length()>=y&&f==StringGrid2-
>Cells[0][i][y]&&x==-1) {
 x=i;

 }
}
 return x;
}
 int find(AnsiString f,leksems*h,int k)
{
int x=-1;
int i=0;
while(h!=0&&x==-1)
{
```

```
 if (f[k]==(h->a)[k]) {
 x=i;
 }
 i++;
 h=h->next;
}
 return x;
}
int find1(AnsiString f,leksems*h)
{
int x=-1;
int i=0;
while(h!=0&&x==-1)
{
 if (f==(h->a)) {
 x=i;
 }
 i++;
 h=h->next;
}
 return x;
}
int t(double num,TStringGrid*StringGrid3)
{
 int y=0;
 int k=num;
 double r=num;
  while(k!=0)
  {
         y+=k%10;
         k=k/10;
  }
   int c=5;
 int i=0;
 if(y<0)
 {y=-y;
 y+=10;}
```

```
  while(StringGrid3->Cells[0][y+c*i]!=""&&y+c*i<=StringGrid3-
>RowCount-1&&StringGrid3->Cells[0][y+c*i]!=num)
  i++;
  if(y+c*i>StringGrid3->RowCount-1)
  StringGrid3->RowCount=y+c*i+1;
  StringGrid3->Cells[0][y+c*i]=num;
  return (y+c*i);
}
void
toleksems(leksems**a,leksan*k,TStringGrid*StringGrid1,TStringGrid
*StringGrid2,TStringGrid*StringGrid3,TStringGrid*StringGrid4,bool
z)
{
 while(k!=0)
 {
  int g=k->a;
  if(g==1)
  (*a)->add((a),(StringGrid1->Cells[0][k->b]),0,0);
  if(g==3)
 { if(z==0)
  (*a)->add((a),"n",0,1);
  else
  (*a)->add((a),(StringGrid3->Cells[0][k->b]),0,1);}
  if(g==4)
  {  if(z==0)
  (*a)->add((a),"a",1,0);
  else
  {(*a)->add((a),StringGrid4->Cells[0][k->b],1,0);
  }
  if(g==2)
  {
  AnsiString h=StringGrid2->Cells[0][k->b];
 if(h=="("||h==")"||h=="{"||h=="}"||h=="="||h==";"||h==","||h=="
:"||z==1)
  (*a)->add((a),h,0,0);
  else (*a)->add((a),"z",0,0);
  }
         k=k->next;
```

```
  }
}
class stack
{ public:
bool eq;
bool nonterm;
bool num;
 AnsiString f;
 stack*next;
 public:
   void add(stack**,AnsiString,bool,bool);
   void add1(stack**,AnsiString,bool);
   void del(stack**);
   void pr(stack*);
 AnsiString forms(stack*);
};
void stack::add(stack**a,AnsiString f,bool z,bool z1)
{
 stack*b=new stack();
 b->nonterm=z;
 b->eq=z1;
 b->f=f;
 b->num=0;
 b->next=0;
 if((*a)==0)
 (*a)=b;
 else
 {b->next=(*a);
 (*a)=b;}
}
 void stack::add1(stack**a,AnsiString f,bool z)
{
 stack*b=new stack();
 b->nonterm=0;
 b->eq=0;
    b->f=f;
 b->num=z;
 b->next=0;
```

```
 if((*a)==0)
 (*a)=b;
 else
 {b->next=(*a);
 (*a)=b;}
}
AnsiString stack::forms(stack*a)
{
AnsiString f="";
while(a!=0)
{
 f.Insert(a->f,1);
 f.Insert(" ",1);
 a=a->next;
}
  return f;
}
 void stack::del(stack**a)
{
 stack*b=(*a);
 (*a)=(*a)->next;
  delete b;
}
 void stack::pr(stack*a)
{
while(a!=0)
{
 ShowMessage(a->f);
 a=a->next;
}
}
class rules
{ public:
int x;
 rules*next;
 public:
   void add(rules**,int);
   void del(rules**);
```

```
   void pr(rules*);
   AnsiString forms(rules*);
};
void rules::add(rules**a,int x)
{
if ((*a)==0) {
 (*a)=new rules();
 (*a)->x=x;
 return;
 }
 rules*c=(*a);
 while(c->next!=0)
 c=c->next;
 rules*d=new rules();
 d->x=x;
 c->next=d;
}
 void rules::del(rules**a)
{
  rules*b=(*a);
  (*a)=(*a)->next;
   delete b;
}
 void rules::pr(rules*a)
{
while(a!=0)
{
  ShowMessage(a->x);
  a=a->next;
}
}
 AnsiString rules::forms(rules*a)
{
AnsiString f="";
if(a==0)
f="k";
while(a!=0)
{
```

```
 f+=a->x;
 f+=" ";
 a=a->next;
}
  return f;
}
stack*a;
rules*r;
 int index1(AnsiString f,TStringGrid*StringGrid1)
 { int i=0;
  while(StringGrid1->Cells[0][i]!=f)
  {
  i++;
   }
   return i;
 }
        int index2(AnsiString f,TStringGrid*StringGrid1)
 { int i=0;
  while(StringGrid1->Cells[i][0]!=f)
  {i++;
   }
   return i;
 }
//---------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
 : TForm(Owner)
{
}
//---------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
StringGrid1->Cells[0][0]="begin";
   StringGrid1->Cells[0][1]="end";
 StringGrid1->Cells[0][2]="dim";
 StringGrid1->Cells[0][3]="integer";
 StringGrid1->Cells[0][4]="real";
 StringGrid1->Cells[0][5]="boolean";
 StringGrid1->Cells[0][6]="as";
```

```
  StringGrid1->Cells[0][7]="switch";
  StringGrid1->Cells[0][8]="case";
StringGrid1->Cells[0][9]="for";
  StringGrid1->Cells[0][10]="while";
StringGrid1->Cells[0][11]="do";
StringGrid1->Cells[0][12]="read";
  StringGrid1->Cells[0][13]="writeln";
  keywords->add(&keywords,"begin",0,0);
  keywords->add(&keywords,"end",0,0);
  keywords->add(&keywords,"dim",0,0);
  keywords->add(&keywords,"integer",0,0);
  keywords->add(&keywords,"real",0,0);
  keywords->add(&keywords,"boolean",0,0);
  keywords->add(&keywords,"as",0,0);
  keywords->add(&keywords,"switch",0,0);
  keywords->add(&keywords,"case",0,0);
  keywords->add(&keywords,"for",0,0);
  keywords->add(&keywords,"while",0,0);
  keywords->add(&keywords,"do",0,0);
  keywords->add(&keywords,"read",0,0);
  keywords->add(&keywords,"writeln",0,0);
StringGrid2->Cells[0][0]="{";
  StringGrid2->Cells[0][1]="}";
StringGrid2->Cells[0][2]="=";
StringGrid2->Cells[0][3]="(";
StringGrid2->Cells[0][4]=")";
StringGrid2->Cells[0][5]="+";
StringGrid2->Cells[0][6]="-";
  StringGrid2->Cells[0][7]="*";
  StringGrid2->Cells[0][8]="/";
StringGrid2->Cells[0][9]="%";
  StringGrid2->Cells[0][10]="&&";
StringGrid2->Cells[0][11]="||";
StringGrid2->Cells[0][12]=",";
StringGrid2->Cells[0][13]="<";
StringGrid2->Cells[0][14]=">";
StringGrid2->Cells[0][15]="<=";
StringGrid2->Cells[0][16]=">=";
```

```
 StringGrid2->Cells[0][17]=";";
 StringGrid2->Cells[0][18]=":";
 StringGrid2->Cells[0][19]="==";
   dels->add(&dels,"{",0,0);
   dels->add(&dels,"}",0,0);
   dels->add(&dels,"=",0,0);
   dels->add(&dels,"(",0,0);
   dels->add(&dels,")",0,0);
   dels->add(&dels,"+",0,0);
   dels->add(&dels,"-",0,0);
   dels->add(&dels,"*",0,0);
   dels->add(&dels,"/",0,0);
   dels->add(&dels,"%",0,0);
   dels->add(&dels,"&&",0,0);
   dels->add(&dels,"||",0,0);
   dels->add(&dels,",",0,0);
   dels->add(&dels,"<",0,0);
   dels->add(&dels,">",0,0);
   dels->add(&dels,"<=",0,0);
   dels->add(&dels,">=",0,0);
   dels->add(&dels,";",0,0);
   dels->add(&dels,":",0,0);
   dels->add(&dels,"==",0,0);
 }
//---------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 for(int i=0;i<StringGrid3->RowCount;i++)
  StringGrid3->Cells[0][i]="";
  for(int i=0;i<StringGrid4->RowCount;i++)
  StringGrid4->Cells[0][i]="";
Memo2->Text="";
 f=0;
leks s=beg;
leks s1;
int sign=1;
AnsiString u="";
AnsiString buf="";
```

```
AnsiString buf1="";
 int h=0;
double num=0;
 bool z=0;
 u=Memo1->Text;
int i=1;
  u.Insert(' ',u.Length()+1);
while(i<=u.Length())
{ z=0;
if(u[i]=='\r')
{i++;
continue;
}
if(s==ign)
{      if(u[i]=='*')
    s=ign1;
    else
    {f->add(&f,2,find('/',StringGrid2,1));
    s=beg;}
    buf="";
}
 if(s==ign1)
{
i++;
while(i<=u.Length()&&u[i]!='*')
i++;
  i++;
if(i>u.Length())
{
 AnsiString h="Незавершённый комментарий";
 Memo2->Text=h;
 s=er;
 return;
}
  s=ign2;
}
if(s==ign2)
{
```

```
if(u[i]=='/')
  {s=beg;
  i++;}
  else s=ign1;
}}
 if(s==sep1||s==sep2||s==sep)
{
   if(s!=sep)
{ if((u[i]=='&'&&s==sep1)||(u[i]=='|'&&s==sep2))
{
 buf+=u[i];
 f->add(&f,2,find1(buf,dels));
 buf="";
 s=beg;
 i++;
}
   else
   {
  AnsiString f="Неверный разделитель ";
  f+=buf;
  f+=u[i];
 Memo2->Text=f;
 s=er;
 return;
  }}
 if(s==sep)
 {
        if(u[i]=='=')
        {buf+=u[i];
        i++;
        }
        f->add(&f,2,find1(buf,dels));
        buf="";
        s=beg;
 }
}
if(s==beg&&(buf=="+"||buf=="-"))
{if(u[i]>=48&&u[i]<=57)
```

```
{
  if(buf=="+")
  sign=1;
  else sign=-1;
}
else f->add(&f,2,find(buf[1],StringGrid2,1));
buf="";}
if(((u[i]>=65&&u[i]<=90)||(u[i]>=97&&u[i]<=122)))
{z=1;
if((s==beg||s==beg1))
s=ident;
}
if(u[i]>=48&&u[i]<=57)
{z=1;
if(s==beg||s==beg1)
s=number;
}
int c=find(u[i],dels,1);
if((s==beg&&s==beg1)&&c==-1&&z==0&&u[i]!='
'&&u[i]!='\n'&&u[i]!='.')
{  AnsiString j="Неверный символ ";
j+=u[i];
Memo2->Text=j;
 s=er;
 return ;
}
if                                                    ((c==-
1&&(s==beg||s==beg1)&&u[i]!=32&&u[i]!='\n')||(u[i]=='.'&&s!=numbe
r)) {
AnsiString f="Неверный разделитель ";
f+=u[i];
 Memo2->Text=f;
 s=er;
 return;
}
  if(s==number)
{
if((u[i]<48||u[i]>57)&&u[i]!='.'&&c==-1&&u[i]!=32&&u[i]!='\n')
```

```
 { buf+=u[i];
 AnsiString k="Неверная цифра ";
 k+=u[i];
  Memo2->Text=k;
  s=er;
  return; }
}
   if(s==number2)
{
if((u[i]<48||u[i]>57)&&c==-1&&u[i]!=32&&u[i]!='\n')
 {   buf+=u[i];
 AnsiString k="Неверная цифра ";
 k+=u[i];
  Memo2->Text=k;
  s=er;
  return; }
}
 if(buf!=""&&(u[i]==32||u[i]=='\n'||i==u.Length()||c!=-
1||(s==number&&u[i]=='.'))&&(s==ident||s==number2||s==number))
 {if(s==ident)
{ int d=analyzeres(buf,keywords);
   if(d==-1)
{h=idents(buf,StringGrid4,Memo2);
if(h==-1)
{s=er;
AnsiString y="Неверный идентификатор ";
y+=buf;
return;}
else
 {f->add(&f,4,h);
 s=beg1;
 buf="";
 }
}
else
s=beg;
buf="";}
```

```
  if((u[i]==32||u[i]=='\n'||u[i]=='.'||i==u.Length()||c!=-
1)&&s==number)
  {for(int u=1;u<=buf.Length();u++)
   {
     num*=10;
     num+=buf[u]-48;
   }
  buf="";
  num*=sign;
  if(u[i]!='.')
  {
  f->add(&f,3,t(num,StringGrid3));
  s=beg1;
  num=0;
  sign=1;}
 else
  s=number2;}
if(s==number2&&(u[i]==32||u[i]=='\n'||i==u.Length()||c!=-1))
{
int p=-1;
 if(sign==-1)
 num=-num;
  for(int u=1;u<=buf.Length();u++)
 {
 num+=(buf[u]-48)*pow(10,p);
 p--;}
  if(sign==-1)
 num=-num;
  sign=1;
 s=beg;
 sign=1;
 if(i>u.Length())
  {
 u.Insert(' ',u.Length()+1);
  }
 f->add(&f,3,t(num,StringGrid3));
 buf="";
 num=0;
```

```
      s=beg1;
     }
     }
if(s!=ign1&&s!=ign1&&i<=u.Length()&&u[i]!=32&&u[i]!='\n'&&u[i]!='.')
     {
     buf+=u[i];
     }
      if(u[i]=='/')
     {s=ign;
     }
      if(c!=-1)
      { if(s==beg1||(s==beg&&buf!='+'&&buf!='-'))
 if(u[i]!='&'&&u[i]!='|'&&u[i]!='<'&&u[i]!='>'&&u[i]!='=')
       {f->add(&f,2,c);
       buf="";
       s=beg;}
       else
       if(u[i]=='&')
       s=sep1;
       else if(u[i]=='|')
       s=sep2;
       else
       {s=sep;         }
      }i++;}
      s=en;
     f->print(f,Memo2);
     }
     //---------------------------------------------------------
     void __fastcall TForm1::Button2Click(TObject *Sender)
     {
     b=0;
     a=0;
     r=0;
     Memo3->Text="";
     Memo4->Text="";
     AnsiString f2,f3,x;
     int i=0;
     OpenDialog2->Execute();
```

```
 x=OpenDialog2->FileName;
ifstream f5(x.c_str());
 f2="";
 f5>>f2;
 f3="";
 while(f2!="")
{ f3=f2.SubString(1,f2.Pos('\n')-1 ) ;
 d[i]=f3;
  f2.Delete(1,f2.Pos('\n'));  i++;
}
toleksems(&b,f,StringGrid1,StringGrid2,StringGrid3,StringGrid4,0)
;
for(int i=0;i<36;i++)
Memo4->Lines->Add(d[i]);
  AnsiString      h=d[0].SubString(d[0].Pos('>')+1,d[0].Length()-
d[0].Pos('>'));
  AnsiString h1=d[0].SubString(1,d[0].Pos('-')-1);
   bool k=1;
 bool fn=0;
 bool z1=0;
 a->add(&a,"/1",0,0);
 b->add (&b,"/0",0,0);
   while(k){
 AnsiString f1=b->forms(b);
 AnsiString f2=a->forms(a);
 AnsiString f3=r->forms(r);
 Memo3->Lines->Add(f1+"|"+f2+"|"+f3);
AnsiString h;
  AnsiString h1;
 {
       if(b->a=="/0")
   {  bool k2=0;
    stack*v=a;
    while(v!=0)
   { if(v->nonterm==0&&v->f!="/1")
    k2=1;
    v=v->next;
   }
```

```
     if(k2==0)
     {k=0;
     break;}
    }
  stack*m;
    if(a->nonterm==1)
  m=a->next;
  else
  m=a;
  int i=index1(m->f,Form2->StringGrid1);
  int j=index2(b->a,Form2->StringGrid1);
  if(Form2->StringGrid1->Cells[j][i]=="<"||Form2->StringGrid1-
>Cells[j][i]=="=")
{
 a->add(&a,b->a,0,0);
 b->del(&b);
}
 if(Form2->StringGrid1->Cells[j][i]=="0")
 {
 Memo3->Lines->Add("Между символами нет предшествования");
 return;
 }
  if(Form2->StringGrid1->Cells[j][i]==">")
 {
  stack*c=a;
  bool z=1;
   stack*c1=c->next;
  while(z==1)
 {
 z=0;
{
z1=1;
 if(h!="")
 h.Insert(" ",1);
 h.Insert(c->f,1);
 if(c->nonterm==0)
 {
        fn=1;
```

```
 }
 if(c->nonterm==1)
 {
 c=c->next;
   a->del(&a);
 }
 if(fn==0)
 {if(h!="")
 h.Insert(" ",1);
 h.Insert(c->f,1);
  fn=1;}
 h1=c->f;
  c=c->next;
 c1=c->next;
 a->del(&a);
  }
  if(c->nonterm==1&&z1==1)
 {
 z=1;
if(h!="")
 h.Insert(" ",1);
 h.Insert(c->f,1);
  c=c->next;
  c1=c->next;
 a->del(&a);
 z1=0;
  }
  int i=index1(c->f,Form2->StringGrid1);
  int j=index2(h1,Form2->StringGrid1);
  if(Form2->StringGrid1->Cells[j][i]=="=")
 { z1=1;
 z=1;
 h1="";
 }
 else
 {z=0;
  }
 }
```

```
 if(Form2->StringGrid1->Cells[j][i]!="=")
 {  bool found=0;
   for(int l=0;l<36;l++)
{
AnsiString        s=d[l].SubString(d[l].Pos('>')+1,d[l].Length()-
d[l].Pos('>'));
  AnsiString s1=d[l].SubString(1,d[l].Pos('-')-1);
if(s==h)
{a->add(&a,s1,1,0);
 found=1;
 r->add(&r,l+1);
 h="";
 break;
}
 }
 if(found==0)
  {
 Memo3->Lines->Add("Не найдено правило");
 return;
 }
 }
 z=0;
 }
 z1=0;
 fn=0;
 h="";
 } }
Memo3->Lines->Add("Принято");
}
//---------------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{
AnsiString f2,f3,x;
int i=0;
OpenDialog2->Execute();
 x=OpenDialog2->FileName;
 ofstream f5(x.c_str());
 f2=Memo1->Text;
```

```
 f5<<f2;
}
//---------------------------------------------------------------
void __fastcall TForm1::Button4Click(TObject *Sender)
{AnsiString f2,f3,x;
int i=0;
OpenDialog2->Execute();
 x=OpenDialog2->FileName;
 ifstream f5(x.c_str());
 f5>>f2;
 Memo1->Text=f2;
}
//---------------------------------------------------------------
void __fastcall TForm1::Button5Click(TObject *Sender)
{
leksems*s=0;
leksems*s1=0;
toleksems(&s,f,StringGrid1,StringGrid2,StringGrid3,StringGrid4,1)
;
s->print(s,Memo5);
leksems*d=s;
while(d!=0)
{
leksems*t;
if(d->a=="dim")
{t=d->next;
while(t->a!=";")
{
  if(t->id==1)
  {    if(s1->find(s1,t->a)==0)
  {
  s1->add(&s1,t->a,1,0);
  }
         else
         {
           AnsiString f="Переопределение переменной ";
         f+=t->a;
                Memo5->Text=f;
```

```
                return;
          }
  }
  t=t->next;
}
d=t;
}
else
if(d->id==1)
 if(s1->find(s1,d->a)==0)
 {
  AnsiString f="Использована необьявленная переменная ";
         f+=d->a;
                Memo5->Text=f;
                return;
 }
d=d->next;
}
  Memo5->Text="Принято";
;
}
//---------------------------------------------------------
void __fastcall TForm1::Button6Click(TObject *Sender)
{Memo6->Text="";
leksems*s=0;
leksems*s1=0;
toleksems(&s,f,StringGrid1,StringGrid2,StringGrid3,StringGrid4,1)
;
stack *v=0;
int PC =0;
int PM=0;
int y=0;
int m=0;
int h=0;
AnsiString d="";
 AnsiString f1;
 AnsiString f2;
 AnsiString f3;
```

```
while(s!=0)
{AnsiString f1=s->forms(s);
 AnsiString f2=v->forms(v);
 AnsiString f3=s1->forms(s1);
 Memo6->Lines->Add(f1+"|"+f2+"|"+f3);
 if(s->a=="as")
 {
   s1->add(&s1,"&",0,0);
 }if(s->id==1||s->num==1)
{  d=s->a;
if(s->num==1&&d[1]=='-')
{h=1;
d.Delete(1,1);
}
 if(h!=2)
s1->add(&s1,d,0,0);
else s1->add(&s1,"=",0,0);
if(h==1)
{s1->add(&s1,"@",0,0);
h=0;}
 s=s->next;
 d="";
}
  else
        if(s->a=="while")
 {
     v->add1(&v,s->a,0);
    AnsiString p="m"+IntToStr(m);
    s1->add(&s1,p,0,0);
    v->add1(&v,IntToStr(m),1);
        m++;
        s=s->next;
 }
   else
  if(v==0&&s->a!="end")
  {v->add1(&v,s->a,0);
        s=s->next;
  }
```

```
else
if(s->a=="("&&v!=0&&v->f=="read")
{
  s=s->next;
}
  else
  if(s->a==")")
 {
 while(v->f!="("&&v->f!="read"&&v->f!="writeln")
  {
      s1->add(&s1,v->f,0,0);
      v=v->next;
  }
        if(v->f!="read")
       v=v->next;
       s=s->next;
 }
  else
  if(s->a==";")
{
 while(v!=0&&v->f!="begin"/*&&v->f!="as"/*&&v->num==0*/)
  {
      if(v->f=="read")
      {s1->add(&s1,"&",0,0);
      s1->add(&s1,v->f,0,0);
      v=v->next;
      }
  if(v!=0)
  {    if(v->f=="as")
  {
      s1->add(&s1,"=",0,0);
      v=v->next;
  }
        if (v!=0&&v->num!=0)
        {
               int i1=StrToInt(v->f);
               v=v->next;
               int i2=StrToInt(v->f);
```

```
                AnsiString p2="m"+IntToStr(i2);
                v=v->next;
                s1->add(&s1,"goto",0,0);
                s1->add(&s1,p2,0,0);
                 AnsiString p1="m"+IntToStr(i1);
                s1->add(&s1,p1,0,0);
                v=v->next;
        }
        else
 {
         if(v!=0&&v->f!="begin")
 {
 s1->add(&s1,v->f,0,0);
     v=v->next;
 }}}}
      s=s->next;
}
else
if(s->a=="end")
{
while(v!=0/*&&v->f!="as")
 {
     if(v->f=="read"}
     {s1->add(&s1,"&",0,0);
     s1->add(&s1,v->f,0,0);
     v=v->next;
     }
 if(v!=0)
 {    if(v->f=="as")
 {
     s1->add(&s1,"=",0,0);
     v=v->next;
 }
       if (v!=0&&v->num!=0)
       {
              int i1=StrToInt(v->f);
              v=v->next;
              int i2=StrToInt(v->f);
```

```
                AnsiString p2="m"+IntToStr(i2);
                v=v->next;
                s1->add(&s1,"goto",0,0);
                s1->add(&s1,p2,0,0);
                 AnsiString p1="m"+IntToStr(i1);
                s1->add(&s1,p1,0,0);
                v=v->next;
        }
        else
 {
         if(v!=0)
 {
 if(v->f!="begin")
 s1->add(&s1,v->f,0,0);
     v=v->next;
 }
 }
 }
  }
      s=s->next;
      if(s->a==";")
     s=s->next;
}
else
if(s->a==",")
{
while(v->f!="writeln"&&v->f!="read")
 {
     if(v->f!="("/*&&v->f!="writeln"*/)
     {
     s1->add(&s1,v->f,0,0);
     }
 if(v->f!="read"&&v->f!="writeln")
     v=v->next;
 }
 if(v->f=="writeln"||v->f=="read")
 {  if(v->f=="read")
```

```
      s1->add(&s1,"&",0,0);
  if(s->a=="("||s->a==")")
  s=s->next;
      s1->add(&s1,v->f,0,0);
  }
       s=s->next;
 }
else
{
 bool z=1;
while(z==1)
{
int r=index2(s->a,Form3->StringGrid1);
stack*k=v;
while(k!=0&&k->num==1)
k=k->next;
 int r1=index2(k->f,Form3->StringGrid1);
  PC=StrToInt(Form3->StringGrid1->Cells[r][1]);
  PM=StrToInt(Form3->StringGrid1->Cells[r1][2]);
 if(PC>PM)
 {
 z=0;
 v->add1(&v,s->a,0);
 s=s->next;
  }
 else
 {
        if(k->f=="while")
        {
         int z=StrToInt(v->f);
         s1->add(&s1,"!",0,0);
        s1->add(&s1,"goto",0,0);
  AnsiString p="m"+IntToStr(m);
         int g=m;
             m++;
  s1->add(&s1,p,0,0);
      v=v->next;
        v=v->next;
```

```
            v->add1(&v,s->a,0);
            v->add1(&v,IntToStr(z),1);
            v->add1(&v,IntToStr(g),1);
            z=0;
            s=s->next;
            break;
            }
            else
    {{if(v->f!="("&&v->f!="begin"&&v->f!=";"&&v->num==0&&v-
>f!="do")
            s1->add(&s1,v->f,0,0); }
            z=1;
            v=v->next;
            if(v==0)
            z=0;
    }
    }
   }
   }
}
   while(v!=0)
   {
     s1->add(&s1,v->f,0,0);
         v=v->next;
   }
 s1->print1(s1,Memo6);
;
}
//-------------------------------------------------------------
------------
```