МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
"Ижевский государственный технический университет имени М.Т.Калашникова"
(ФГБОУ ВПО «ИжГТУ имени М.Т.Калашникова»)

Кучуганов В.Н., Касимов Д.Р.

МАТЕМАТИЧЕСКАЯ ЛИНГВИСТИКА
МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №1
«РАЗРАБОТКА ЛЕКСИЧЕСКОГО АНАЛИЗАТОРА»

Рекомендовано учебно-методическим советом ФГБОУ ВПО «ИжГТУ имени М.Т. Калашникова» для использования в учебном процессе в качестве
элемента ЭУМКД для студентов обучающихся по направлению 230100.62 «Информатика и вычислительная техника», профилям «Автоматизированные системы обработки информации и управления», «Системы автоматизированного проектирования» при изучении дисциплин «Математическая лингвистика», «Лингвистическое обеспечение САПР»

Ижевск 2013

Составители: Кучуганов Валерий Никонорович, доктор технических наук, профессор
Касимов Денис Рашидович, ассистент

УДК 681.3

Математическая лингвистика: методические указания к выполнению лабораторной работы №1 «Разработка лексического анализатора» по курсам «Математическая лингвистика», «Лингвистическое обеспечение САПР» профилей «Автоматизированные системы обработки информации и управления», «Системы автоматизированного проектирования» направления 230100.62 «Информатика и вычислительная техника».
Составители: Кучуганов В.Н., Касимов Д.Р., Ижевский государственный технический университет имени М.Т. Калашникова. Ижевск, 2013. – 23 с.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Первой фазой работы лингвистического процессора является лексический анализ, то есть группирование строк литер, обозначающих идентификаторы, константы или слова языка и т.д., в единые символы (лексемы). Этот процесс может идти параллельно с другими фазами обработки текста (например, в однопроходных лингвистических процессорах). Однако, в любом случае, при описании конструкции лингвистического процессора и его построения удобно представлять лексический анализ как самостоятельную фазу.

Блок сканирования (сканер) должен выдавать каждую лексему, устанавливая ее принадлежность тому или иному классу. Выбор классов зависит от особенностей транслируемого языка. Часто выделяют классы имен переменных, констант, ключевых слов, арифметических и логических операций ("+", "–", "*", "/" и т.д.), специальных символов ("=", ";" и т.д.). Все эти строки можно генерировать с помощью регулярных выражений. Например, вещественные числа можно генерировать посредством регулярного выражения (+ | –) d d*.d*, где d обозначает любую цифру. Из выражения видно, что вещественное число состоит из следующих компонентов, расположенных именно в таком порядке:

1) возможного знака;
2) последовательности из одной или более цифр;
3) десятичной точки;
4) последовательности из нуля или более цифр.

Лексический анализатор, наряду с разбиением исходного потока символов на лексемы, может включать в исходный текст дополнительную информацию или исключать из него строки символов. Примером дополнительно включаемой информации являются номера строк. Если их не включить, то информация о том, в какой строке исходного текста находилась та или иная лексема, будет, в случае выполняющего сканирование в отдельном проходе лингвистического процессора, утеряна после лексического анализа. Однако для диагностики на фазе синтаксического анализа необходимо иметь возможность ссылаться на ошибки в программе с указанием номеров строк оригинального исходного текста. С другой стороны, комментарии включаются в текст программы или описания объекта проектирования только для предоставления человеку дополнительных пояснений. Они никак не влияют на генерируемый в дальнейшем код, и лексический анализатор обычно их просто исключает.

Цель описанной ниже лабораторной работы – ознакомиться с теоретическими и практическими основами построения блока лексического анализа лингвистического процессора.

1. ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ №1

Тема работы: «Разработка лексического анализатора».

Цель работы: ознакомиться с теоретическими и практическими основами построения блока лексического анализа лингвистического процессора.

Используемые программные средства: система программирования Delphi 7.0 или выше; графический редактор Microsoft Visio.

Задание по лабораторной работе заключается в разработке лексического анализатора для предложенного варианта слов. На вход подается многострочный текст, содержащий два вида слов, следующих в произвольном порядке. Слова одного вида должны быть разделены любым количеством пробелов. Слова разного вида могут записываться слитно или разделяться любым количеством пробелов. Текст может содержать комментарии. Комментарии и пробелы должны пропускаться. Результатом работы программы является диагностическое сообщение о первом неверном слове (если такое встретится), сообщение о правильности поступившего текста.

Содержание отчета:

1) титульный лист;
2) текст задания, включающий вариант задания;
3) по каждому виду слов и комментарию:
   3.1) описание регулярной грамматики;
   3.2) описание недетерминированного конечного автомата (в виде диаграммы и матрицы);
   3.3) описание детерминированного конечного автомата (в виде матрицы и диаграммы);
   3.4) описание объединенного алгоритма (в виде диаграммы);
4) исходный текст программы;
5) результаты тестирования.

Методические рекомендации к лабораторной работе

Обозначения. В формулировке вариантов слов используются регулярные выражения (см. раздел конспекта лекций «Документирование языков»).

Выражение вида (x)* обозначает множество всех строк вида x, xx, xxx, … , включая, пустую строку.

Выражение вида (x)+ обозначает множество всех строк вида x, xx, xxx, … , исключая пустую строку.

Выражение вида | означает «или».

Таким образом, (,), | – это символы метаязыка, не являются частью описываемого языка.

Например, регулярное выражение (111)*110(000)* обозначает слова: 110, 111110, 110000, 111110000, 111111110000, 111110000000, и т.д. Любое слово данного вида содержит обязательную часть 110, необязательный префикс из любого числа повторений 111, необязательный суффикс из любого числа повторений 000.

Регулярное выражение (a|b|c|d)+ обозначает слова: a, b, c, d, aa, ab, ac, ad, ba, bb, bc, bd, ca, cb, cc, cd, da, db, dc, dd, aaa, aab, aac, aad, aba, abb, abc, abd, и т.д.

Внимание! Для второго вида слов дополнительно нужно учитывать ограничение, записанное в третьей колонке таблицы вариантов.

План работы:

1. Составить регулярную грамматику для каждого вида слов и комментария.
2. По каждой грамматике построить конечный автомат.
3. Преобразовать полученные конечные автоматы в детерминированные конечные автоматы, если они такими не являются.
4. Составить объединенный алгоритм лексического анализатора.
5. Написать и отладить программу по алгоритму. Предусмотреть транслитератор и обработчик ошибок в качестве отдельных подпрограмм.

Рекомендации:

Оформить лексический анализатор в виде процедуры или функции или класса. За одно обращение обрабатывает одно очередное слово. Управление потоком слов осуществлять отдельно – цикл, в котором с помощью лексического анализатора пословно читается входной текст до конца или до первой ошибки.

При отладке программы обратить внимание на обработку правильного текста, обработку ошибочного текста, обработку конца входного текста.

Варианты индивидуальных заданий

Варианты заданий приведены в таблице 1.1.

Таблица 1.1. Варианты заданий на лабораторную работу

№	Первое слово	Второе слово	Условие для второго слова	Комментарий
1	(101)*010(011)*	(a|b|c|d)+	Первый символ в слове всегда a	; однострочный комментарий (как в Ассемблере)
2	(101)*011(111)*	(a|b|c|d)+	Первый символ в слове всегда b	# однострочный комментарий (как в PHP)

3	(101)*100(110)*	(a|b|c|d)+	Если начинается с d, то не должно встретиться b	{ многострочный комментарий } (как в Delphi)
4	(101)*110(111)*	(a|b|c|d)+	Первый символ в слове всегда c	// однострочный комментарий (как в Delphi)
5	(110)*000(001)*	(a|b|c|d)+	Нет подстроки ac	(* многостр. комментарий *) (как в Delphi)
6	(110)*010(011)*	(a|b|c|d)+	Если начинается с d, то не должно встретиться c	<!-- многостр. комментарий --> (как в HTML)
7	(110)*011(100)*	(a|b|c|d)+	Не должно заканчиваться dd	' однострочный комментарий (как в Visual Basic)
8	(110)*100(101)*	(a|b|c|d)+	Первый символ в слове всегда d	-- однострочный комментарий (как в Lua)
9	(110)*101(111)*	(a|b|c|d)+	Нет подстроки ad	/* многостр. комментарий */ (как в C++)
10	(110)*111(000)*	(a|b|c|d)+	Не должно заканчиваться da	; однострочный комментарий (как в Ассемблере)
11	(111)*000(001)*	(a|b|c|d)+	Не должно заканчиваться bd	# однострочный комментарий (как в PHP)
12	(111)*010(011)*	(a|b|c|d)+	Нет подстроки aa	{ многострочный комментарий } (как в Delphi)
13	(111)*011(100)*	(a|b|c|d)+	Нет подстроки ab	(* многостр. комментарий *) (как в Delphi)
14	(111)*100(001)*	(a|b|c|d)+	Не должно заканчиваться bc	// однострочный комментарий (как в Delphi)
15	(111)*101(110)*	(a|b|c|d)+	Не должно заканчиваться cb	<!-- многостр. комментарий --> (как в HTML)
16	(111)*110(000)*	(a|b|c|d)+	Первые два символа всегда cd	' однострочный комментарий (как в Visual Basic)
17	(010)*100(000)*	(a|b|c|d)+	Если начинается с b, то не должно встретиться a	-- однострочный комментарий (как в Lua)
18	(010)*101(110)*	(a|b|c|d)+	В порядке обратном алфавитному	/* многостр. комментарий */ (как в C++)
19	(010)*110(111)*	(a|b|c|d)+	Вторые два символа всегда ab	; однострочный комментарий (как в Ассемблере)
20	(011)*000(001)*	(a|b|c|d)+	Вторые два символа всегда ac	# однострочный комментарий (как в PHP)
21	(011)*001(010)*	(a|b|c|d)+	Не должно заканчиваться aa	{ многострочный комментарий } (как в Delphi)
22	(011)*010(100)*	(a|b|c|d)+	Вторые два символа всегда ab	// однострочный комментарий (как в Delphi)
23	(011)*100(101)*	(a|b|c|d)+	Вторые два символа всегда ba	(* многостр. комментарий *) (как в Delphi)
24	(011)*101(110)*	(a|b|c|d)+	Если начинается с d, то не должно встретиться a	<!-- многостр. комментарий --> (как в HTML)
25	(011)*111(000)*	(a|b|c|d)+	Только b и d могут повторяться	' однострочный комментарий (как в Visual Basic)

26	(100)*000(001)*	(a|b|c|d)+	Вторые два символа всегда bc	-- однострочный комментарий (как в Lua)
27	(100)*001(010)*	(a|b|c|d)+	Вторые два символа всегда bd	/* многостр. комментарий */ (как в C++)
28	(100)*010(011)*	(a|b|c|d)+	Не должно заканчиваться cc	; однострочный комментарий (как в Ассемблере)
29	(100)*011(100)*	(a|b|c|d)+	Если начинается с b, то не должно встретиться c	# однострочный комментарий (как в PHP)
30	(100)*101(110)*	(a|b|c|d)+	Нет подстроки bd	{ многострочный комментарий } (как в Delphi)
31	(100)*110(100)*	(a|b|c|d)+	Если начинается с b, то не должно встретиться d	// однострочный комментарий (как в Delphi)
32	(101)*000(001)*	(a|b|c|d)+	Повторение символов может быть только рядом	(* многостр. комментарий *) (как в Delphi)
33	(101)*001(010)*	(a|b|c|d)+	В алфавитном порядке	<!-- многостр. комментарий --> (как в HTML)
34	(000)*001(010)*	(a|b|c|d)+	Первые два символа всегда ab	' однострочный комментарий (как в Visual Basic)
35	(000)*001(100)*	(a|b|c|d)+	Нет подстроки bb	-- однострочный комментарий (как в Lua)
36	(000)*010(011)*	(a|b|c|d)+	Не должно начинаться с ab	/* многостр. комментарий */ (как в C++)
37	(000)*010(100)*	(a|b|c|d)+	Первые два символа всегда ac	; однострочный комментарий (как в Ассемблере)
38	(000)*100(010)*	(a|b|c|d)+	Первые два символа всегда ad	# однострочный комментарий (как в PHP)
39	(000)*101(110)*	(a|b|c|d)+	Первые два символа всегда ba	{ многострочный комментарий } (как в Delphi)
40	(001)*000(010)*	(a|b|c|d)+	Если начинается с a, то не должно встретиться b	// однострочный комментарий (как в Delphi)
41	(001)*010(011)*	(a|b|c|d)+	Первые два символа всегда bc	(* многостр. комментарий *) (как в Delphi)
42	(001)*010(100)*	(a|b|c|d)+	Не должно заканчиваться cd	<!-- многостр. комментарий --> (как в HTML)
43	(001)*010(101)*	(a|b|c|d)+	Первые два символа всегда bd	' однострочный комментарий (как в Visual Basic)
44	(001)*011(000)*	(a|b|c|d)+	Если начинается с a то (a)+	-- однострочный комментарий (как в Lua)
45	(001)*100(101)*	(a|b|c|d)+	Длина не более 4	/* многостр. комментарий */ (как в C++)
46	(001)*101(110)*	(a|b|c|d)+	Первые два символа всегда ca	; однострочный комментарий (как в Ассемблере)
47	(010)*000(001)*	(a|b|c|d)+	Первые два символа всегда cb	# однострочный комментарий (как в PHP)
48	(010)*001(000)*	(a|b|c|d)+	Нет подстроки bc	{ многострочный комментарий } (как в Delphi)

49	(010)*011(100)*	(a|b|c|d)+	Если начинается с d, то не должно встретиться a	// однострочный комментарий (как в Delphi)

Контрольные вопросы:

1. Классификация языков, грамматик, автоматов по Хомскому.
2. Конструирование транслитераторов.
3. Регулярные грамматики и конечные автоматы.
4. Преобразование и оптимизация конечных автоматов.
5. Технология разработки лексических анализаторов.
6. Алгоритмы лексического анализа.
7. Программирование лексического анализатора.
8. Оформление подпрограмм лексического анализа.
9. Документирование регулярных языков. Регулярные выражения.
10. Генераторы лексических анализаторов.
11. Разработка лексики формального языка.

2. РЕГУЛЯРНЫЕ ГРАММАТИКИ

Регулярная грамматика — формальная грамматика типа 3 по иерархии Хомского. Регулярные грамматики определяют в точности все регулярные языки, и поэтому эквивалентны конечным автоматам и регулярным выражениям. Регулярные грамматики являются подмножеством контекстно-свободных.

Регулярная грамматика может быть задана набором правил как левая или правая регулярная грамматика.

Правая регулярная грамматика – все правила могут быть в одной из следующих форм:

$A \to a$

$A \to aB$

$A \to \varepsilon$

Левая регулярная грамматика – все правила могут быть в одной из следующих форм:

$A \to a$

$A \to Ba$

$A \to \varepsilon$

где заглавные буквы (A, B) обозначают нетерминалы из множества N;

строчные буквы (a, b) обозначают терминалы из множества Σ;

ε – пустая строка, т.е. строка длины 0.

Классы правых и левых регулярных грамматик эквивалентны – каждый в отдельности достаточен для задания всех регулярных языков. Любая регулярная грамматика может быть преобразована из левой в правую, и наоборот.

Пример. Правая регулярная грамматика G, заданная $N = \{S, A\}$, $\Sigma = \{a, b, c\}$, P состоит из следующих правил:

$S \rightarrow aS$

$S \rightarrow bA$

$A \rightarrow \varepsilon$

$A \rightarrow cA$

и S является начальным символом. Эта грамматика описывает тот же язык, что и регулярное выражение a*bc*.

3. КОНЕЧНЫЕ АВТОМАТЫ

Конечный автомат (КА) – абстрактный автомат без выходного потока, число возможных состояний которого конечно. Результат работы автомата определяется по его конечному состоянию.

Существуют различные способы задания конечного автомата. Например, конечный автомат может быть задан в виде упорядоченной пятерки:

$$M = (V, Q, q_0, F, \delta),$$

где V – входной алфавит (конечное множество входных символов), из которого формируются входные цепочки, допускаемые конечным автоматом;

Q – множество состояний;

q_0 – начальное состояние ($q_0 \in Q$);

F – множество заключительных состояний ($F \subset Q$);

δ – функция переходов, определяющая для упорядоченной пары <состояние, входной символ или пустая цепочка> множество всех состояний, в которые из данного состояния возможен переход по данному входному символу или пустой цепочке.

Конечный автомат начинает работу в состоянии q_0, считывая по одному символу входной цепочки. Считанный символ переводит автомат в новое состояние в соответствии с функцией переходов. Читая входную цепочку x и делая один такт за другим, автомат, после того как он прочитает последнюю букву цепочки, окажется в каком-то состоянии q'. Если это состояние является заключительным, то говорят, что автомат допустил цепочку x.

Способы описания конечного автомата:

1. Диаграмма состояний (или иногда граф переходов) – графическое представление множества состояний и функции переходов. Представляет собой размеченный ориентированный граф, вершины которого – состояния КА, дуги – переходы из одного состояния в другое, а метки дуг – символы, по которым осуществляется переход из одного состояния в другое. Если переход из состояния q_1 в q_2 может быть осуществлен по одному из нескольких символов, то все они должны быть надписаны над дугой диаграммы.

2. Таблица переходов – табличное представление функции δ. Обычно в такой таблице каждой строке соответствует одно состояние, а столбцу – один допустимый входной символ. В ячейке на пересечении строки и столбца записывается состояние, в которое должен перейти автомат, если в данном состоянии он считал данный входной символ.

Конечные автоматы подразделяются на детерминированные и недетерминированные. Детерминированным конечным автоматом (ДКА) называется такой автомат, в котором нет дуг с меткой ε и из любого состояния по любому символу возможен переход в точности в одно состояние. Недетерминированный конечный автомат (НКА) является обобщением детерминированного.

4. ПОСТРОЕНИЕ НЕДЕТЕРМИНИРОВАННОГО КОНЕЧНОГО АВТОМАТА

Для построения на основе регулярной грамматики *G* конечного автомата необходимо нетерминальным символам грамматики *N* поставить в соответствие состояния автомата, начальному нетерминалу *S* – начальное состояние автомата *S*, правилам вывода – переходы автомата.

Результатом является таблица или граф переходов недетерминированного конечного автомата.

Полученный автомат является не полностью определенным (частичным), поскольку в таблице переходов есть незаполненные клетки, и, скорее всего, недетерминированным, если в той же таблице есть клетки, содержащие пары состояний.

Чтобы получить полностью определенный автомат, следует ввести дополнительное состояние *Er* (ошибка). Для этого нужно дополнить автомат соответствующей строкой с состоянием *Er* и во все пустые клетки таблицы переходов вписать это состояние *Er*.

Нетрудно убедиться, что построенный автомат допускает все те и только те цепочки символов, которые принадлежат языку *L(G)*, порождаемому грамматикой *G*.

5. СВЕДЕНИЕ НЕДЕТЕРМИНИРОВАННОГО КОНЕЧНОГО АВТОМАТА К ДЕТЕРМИНИРОВАННОМУ

Процедура построения детерминированного автомата A_D, эквивалентного недетерминированному автомату A_N, задается следующими шагами:

1. Пометить первую строку таблицы переходов для A_D множеством начальных состояний автомата A_N. Применить к этому множеству шаг 2.

2. По данному множеству состояний B, помечающему строку таблицы переходов автомата A_D, для которой переходы еще не вычислены, вычислить те состояния автомата A_N, которые могут быть достигнуты из B с помощью каждого входного символа x, и поместить полученные множества состояний в соответствующие ячейки таблицы для автомата A_D. Символически это выражается так: если δ – функция недетерминированных переходов, то функция детерминированных перехода δ' задается формулой $\delta'(B, x) = \{S \mid S \in \delta(T, x), \forall\ T \in B\}$.

3. Для каждого нового множества, порожденного переходами на шаге 2, посмотреть, имеется ли уже в A_D строка, помеченная этим множеством. Если нет, то создать новую строку и пометить ее этим множеством. Если множество уже использовалось как метка, никаких действий не требуется.

4. Если в таблице автомата A_D есть строка, для которой еще не вычислены переходы, вернуться назад и применить к этой строке шаг 2. Если все переходы вычислены, перейти к шагу 5.

5. Пометить строку как допускающее состояние автомата A_D тогда и только тогда, когда она содержит допускающее состояние недетерминированного автомата. В противном случае пометить как отвергающее состояние.

Описанная процедура гарантирует, что детерминированный автомат не содержит недостижимых состояний.

6. ПРИМЕР ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В данной главе приводятся результаты выполнения лабораторной работы при следующих исходных данных:

Первое слово: $(000)^*010(100)^*$.

Второе слово: $(a|b|c|d)^+$.

Условие для второго слова: не должно начинаться с ab.

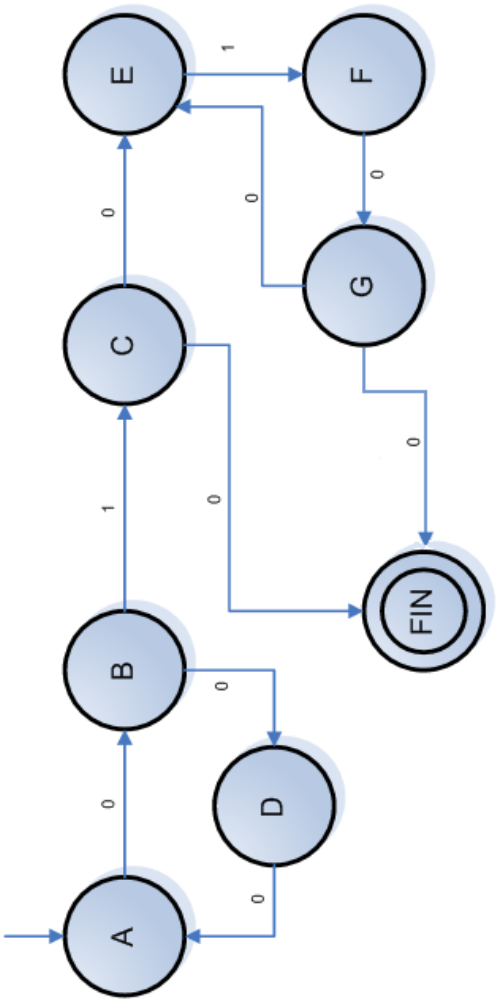
Комментарий: { многострочный комментарий } (как в Delphi).

В таблице 6.1 представлены регулярные грамматики слов и комментария.

Таблица 6.1. Регулярные грамматики

Первого слова	Второго слова	Комментария
A → 0B B → 1C | 0D D → 0A C → 0 | 0E E → 1F F → 0G G → 0 | 0E	A → aC | bB | cB | dB | a | b | c | d B → aB | bB | cB | dB | a | b | c | d C → aB | cB | dB | a | c | d	A → { B B → любой символ кроме } B B → }

На рисунке 6.1. представлен НКА для первого слова.



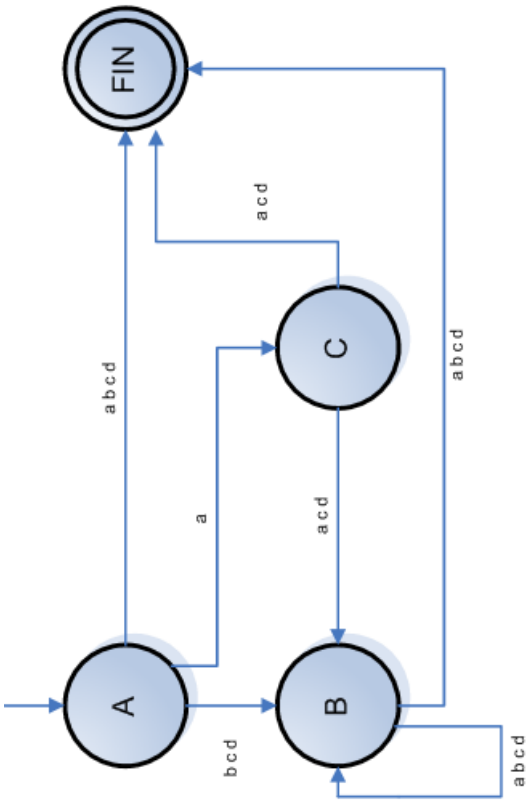
а)

	0	1	*Явл. фин.?*
A	B		*Нет*
B	D	C	*Нет*
C	E, FIN		*Нет*
D	A		*Нет*
E		F	*Нет*
F	G		*Нет*
G	E, FIN		*Нет*
FIN			*Да*

б)

Рисунок 6.1. НКА для первого слова: а) диаграмма; б) матрица

На рисунке 6.2. представлен НКА для второго слова.



а)

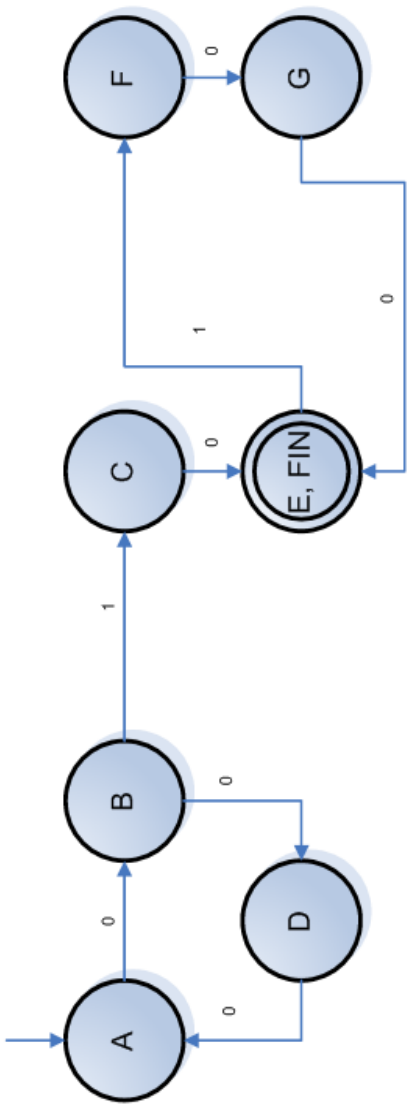
	a	b	c	d	*Явл. фин.?*
A	C, FIN	B, FIN	B, FIN	B, FIN	*Нет*
B	B, FIN	B, FIN	B, FIN	B, FIN	*Нет*
C	B, FIN		B, FIN	B, FIN	*Нет*
FIN					*Да*

б)

Рисунок 6.2. НКА для второго слова: а) диаграмма; б) матрица

Автомат для комментария изначально детерминированный (см. рисунок 6.5).

На рисунке 6.3. представлен ДКА для первого слова.



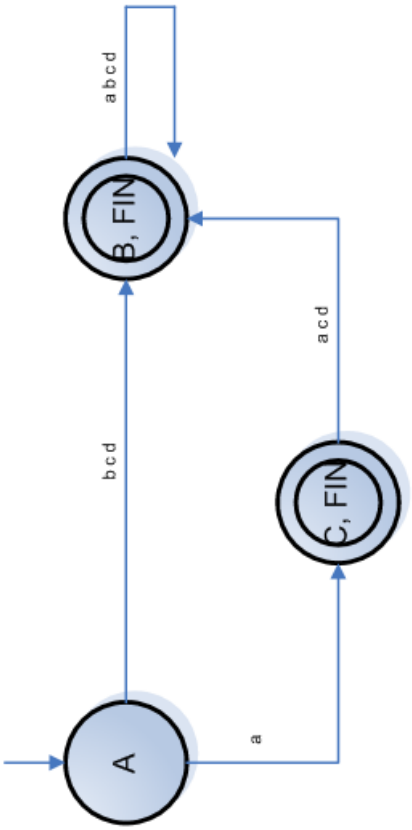
а)

	0	1	*Явл. фин.?*
{A}	{B}		*Нет*
{B}	{D}	{C}	*Нет*
{C}	{E, FIN}		*Нет*
{D}	{A}		*Нет*
{E, FIN}		{F}	*Да*
{F}	{G}		*Нет*
{G}	{E, FIN}		*Нет*

б)

Рисунок 6.3. ДКА для первого слова: а) диаграмма; б) матрица

На рисунке 6.4. представлен ДКА для второго слова.



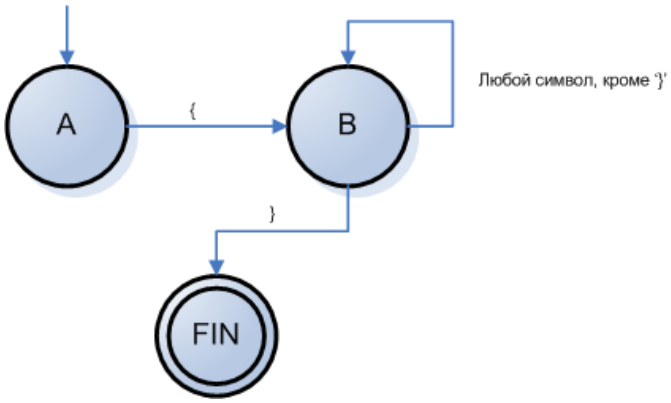
а)

	a	b	c	d	*Явл. фин.?*
{A}	{C, FIN}	{B, FIN}	{B, FIN}	{B, FIN}	*Нет*
{B, FIN}	{B, FIN}	{B, FIN}	{B, FIN}	{B, FIN}	*Да*
{C, FIN}	{B, FIN}		{B, FIN}	{B, FIN}	*Да*

б)

Рисунок 6.4. ДКА для второго слова: а) диаграмма; б) матрица

На рисунке 6.5. представлен ДКА для комментария.



а)

	{	Любой символ, кроме ‘}’	}	*Явл. фин.?*
A	B			*Нет*
B		B	FIN	*Нет*
FIN				*Да*

б)

Рисунок 6.5. ДКА для комментария: а) диаграмма; б) матрица

На рисунке 6.6 представлен объединенный алгоритм лексического анализа.

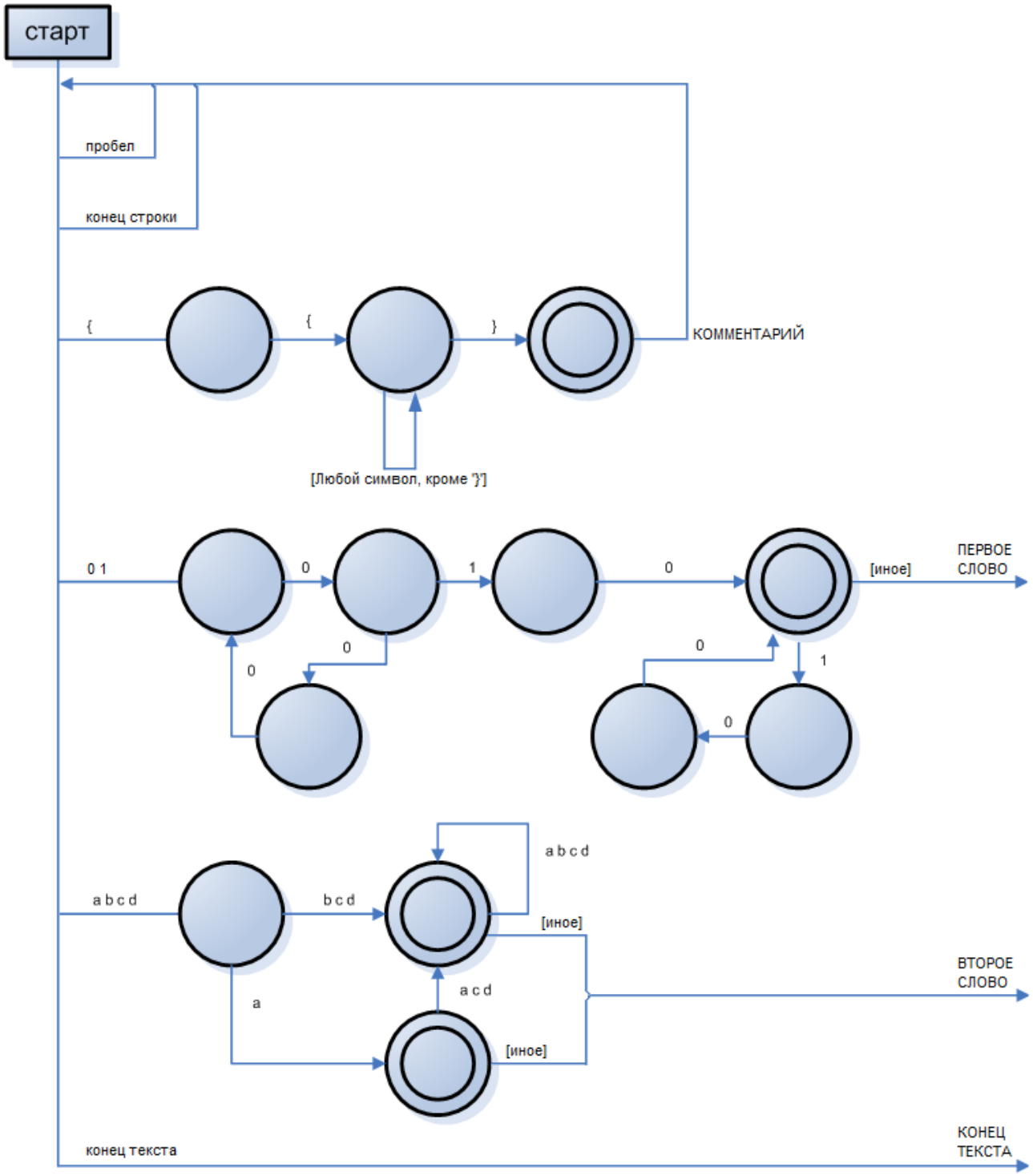
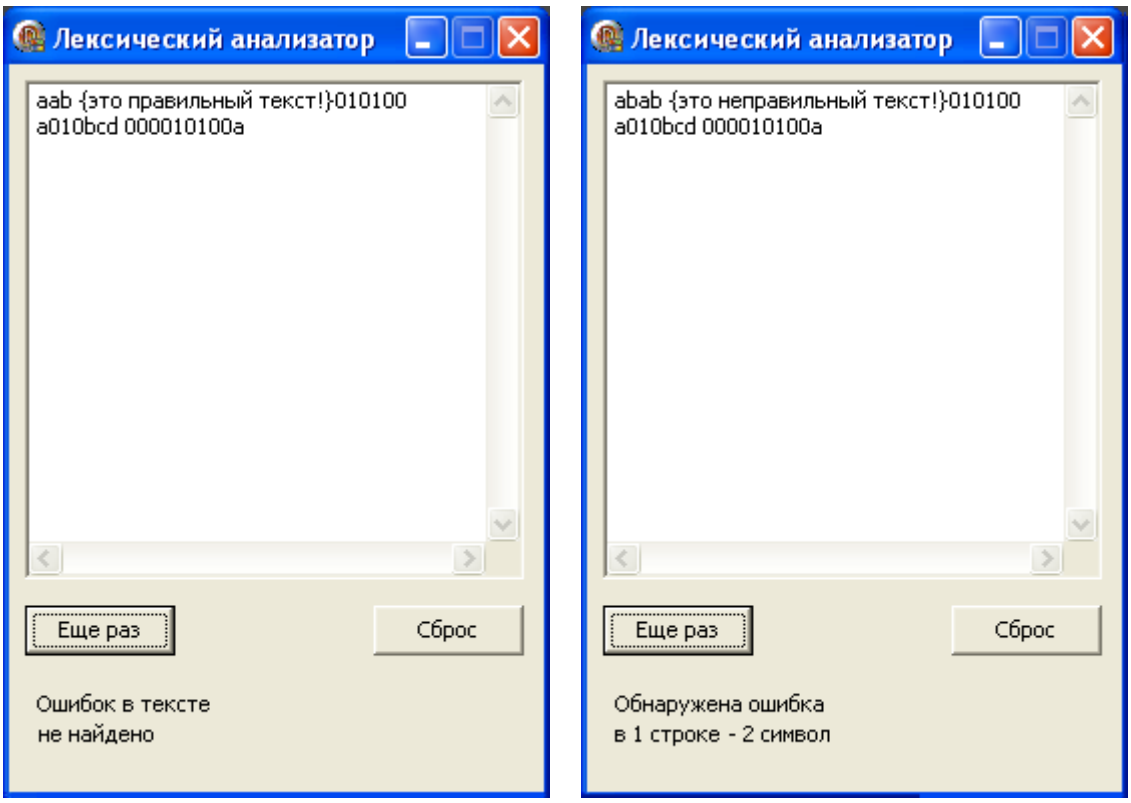


Рисунок 6.6. Объединенный алгоритм лексического анализа

На рисунке 6.7 изображены результаты работы программы при правильном и неправильном введенном тексте.



а) б)

Рисунок 6.7. Результаты работы программы: а) при правильном тексте; б) при неправильном тексте

Ниже представлен исходный текст программы.

```
unit Unit_lexical_analysis;
interface
uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
     Dialogs, StdCtrls;
type
  TfrmMainForm = class(TForm)
    MemInput: TMemo;
    btnRun: TButton;
    lblInfo: TLabel;
    btnReset: TButton;
    procedure btnRunClick(Sender: TObject);
    procedure btnResetClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  end;
var
  frmMainForm: TfrmMainForm;
type
  TLexicalAnalyser = class
    flblHint: TLabel;
    fmemInput: TMemo;
    fintRowNumber: Integer; // номер строки
    fintColNumber: Integer; // номер символа в строке
    fstrWord: String;       // само слово
    fintLinesNumber: Integer;          // общее число строк
    fintSymbolsInLineNumber: Integer; // общее число символов в текущей строке
    fTextEnd: Boolean;                 // маркер конца текста
```

```
    fNewLine: Boolean;                // маркер начала новой строки
    constructor Create(mem: TMemo; lbl: TLabel);
    procedure Transliterator;
    procedure Scaner;
    procedure Error;
  end;

implementation

{$R *.dfm}

var
  LexAn: TLexicalAnalyser;

{=====================================================================}
constructor TLexicalAnalyser.Create(mem: TMemo; lbl: TLabel);
begin
  fmemInput := mem;
  flblHint := lbl;
  fstrWord := '';
  fintRowNumber := 0;
  fintColNumber := 1;
  fTextEnd := false;
  fNewLine := false;
  fintLinesNumber := fmemInput.Lines.Count-1;
  fintSymbolsInLineNumber := length(fmemInput.Lines[0]);
end;
{=====================================================================}
procedure TLexicalAnalyser.Transliterator;
begin
  if fintRowNumber <= fintLinesNumber then
    begin
      // обнаружен конец текста
      if (fintColNumber = fintSymbolsInLineNumber) and
         (fintRowNumber = fintLinesNumber)
      then begin
              fTextEnd := true;
              Exit;
           end;
      // обнаружен конец строки(любой, кроме последней)
      if fintColNumber = fintSymbolsInLineNumber then
      begin
        fintColNumber := 0;
        inc(fintRowNumber);
        fNewLine := true;
        fintSymbolsInLineNumber := length(fmemInput.Lines[fintRowNumber]);
        Transliterator;
        Exit;
      end;

      inc(fintColNumber);
    end;
end;
{=====================================================================}
procedure TLexicalAnalyser.Error;
begin
  flblHint.Caption := 'Обнаружена ошибка в '+ IntToStr(fintRowNumber+1) +
                      ' строке - ' + IntToStr(fintColNumber) + ' символ';
end;
{=====================================================================}
procedure TLexicalAnalyser.Scaner;
label O, A2, B2, C2, D2, EFIN2, F2, G2, T, A1, FIN1, A3;
begin
    O:
```

```
      begin
      if length(fmemInput.Text) = 0 then
      begin flblHint.Caption := 'Введите хоть что-нибудь!'; Exit; end;
      // пропуск пустых строк
      while fintSymbolsInLineNumber = 0 do
      begin
        if fintRowNumber = fintLinesNumber then
        begin
          fTextEnd := true;
          goto T;
        end;
        inc(fintRowNumber);
        fintSymbolsInLineNumber := length(fmemInput.Lines[fintRowNumber]);
      end;
      // пропуск пробелов
      while (fmemInput.Lines[fintRowNumber][fintColNumber] = ' ')
      and not fTextEnd do
      begin
        Transliterator;
        if fintSymbolsInLineNumber = 0 then goto O;
      end;
    end;

    fNewLine := false;

    T:
      if (fTextEnd) and (fstrWord='') then
      begin flblHint.Caption := 'Зачем разводить столько пустого места? Лучше
введите текст!'; Exit; end;
      if (fTextEnd) and (fstrWord <> '') then
      begin flblHint.Caption := 'Ошибок в тексте не найдено'; Exit; end;


  case fmemInput.Lines[fintRowNumber][fintColNumber] of
{буквенная грамматика - начало}
    'a','b','c','d' :
      begin
        if fmemInput.Lines[fintRowNumber][fintColNumber] = 'a' then goto A1
        else goto FIN1;
    A1:
        fstrWord := fstrWord + fmemInput.Lines[fintRowNumber][fintColNumber];
        Transliterator;
        if fNewLine then goto O;
        if fTextEnd then begin flblHint.Caption := 'Ошибок в тексте не найдено';
Exit; end;
        if fmemInput.Lines[fintRowNumber][fintColNumber] = 'b'
        then begin Error; Exit; end;
        if (fmemInput.Lines[fintRowNumber][fintColNumber] = 'a') or
(fmemInput.Lines[fintRowNumber][fintColNumber] = 'c')
        or (fmemInput.Lines[fintRowNumber][fintColNumber] = 'd') then goto FIN1
        else goto O;
FIN1 :
        fstrWord := fstrWord + fmemInput.Lines[fintRowNumber][fintColNumber];
        Transliterator;
        if fNewLine then goto O;
        if fTextEnd then begin flblHint.Caption := 'Ошибок в тексте не найдено';
Exit; end;
        if (fmemInput.Lines[fintRowNumber][fintColNumber] = 'a') or
(fmemInput.Lines[fintRowNumber][fintColNumber] = 'b')
        or (fmemInput.Lines[fintRowNumber][fintColNumber] = 'c') or
(fmemInput.Lines[fintRowNumber][fintColNumber] = 'd')
        then goto FIN1
        else goto O;
      end;
```

```
{буквенная грамматика - конец}
{цифровая грамматика - начало}
    '0','1' :
      begin
    A2 :
        if (fmemInput.Lines[fintRowNumber][fintColNumber] <> '0') or fNewLine
then
        begin Error; Exit; end;
    B2 :
        begin
        fstrWord := fstrWord + fmemInput.Lines[fintRowNumber][fintColNumber];
        Transliterator;
        if fNewLine then
        begin
              flblHint.Caption := 'Обнаружена ошибка! Непредсказуемый переход на
' + IntToStr(fintRowNumber+1) + ' строчку';
              Exit;
        end;
        if fTextEnd then begin Error; Exit; end;
        if fmemInput.Lines[fintRowNumber][fintColNumber] = '1' then goto C2;
        if fmemInput.Lines[fintRowNumber][fintColNumber] = '0' then goto D2
        else begin Error; Exit; end;
        end;
    C2 :
        begin
        fstrWord := fstrWord + fmemInput.Lines[fintRowNumber][fintColNumber];
        Transliterator;
        if fNewLine then
        begin
              flblHint.Caption := 'Обнаружена ошибка! Непредсказуемый переход на
' + IntToStr(fintRowNumber+1) + ' строчку';
              Exit;
        end;
        if fTextEnd then begin Error; Exit; end;
        if fmemInput.Lines[fintRowNumber][fintColNumber] = '0' then goto EFIN2
        else begin Error; Exit; end;
        end;
    D2 :
       begin
       fstrWord := fstrWord + fmemInput.Lines[fintRowNumber][fintColNumber];
       Transliterator;
       if fTextEnd then begin Error; Exit; end;
       if fmemInput.Lines[fintRowNumber][fintColNumber] = '0' then
       begin
            fstrWord := fstrWord +
fmemInput.Lines[fintRowNumber][fintColNumber];
            Transliterator;
            if fNewLine then
            begin
              flblHint.Caption := 'Обнаружена ошибка! Непредсказуемый переход на
' + IntToStr(fintRowNumber+1) + ' строчку';
              Exit;
            end;
            if fTextEnd then begin Error; Exit; end;
            goto A2;
       end
       else begin Error; Exit; end;
       end;
EFIN2 :
       begin
       fstrWord := fstrWord + fmemInput.Lines[fintRowNumber][fintColNumber];
       Transliterator;
       if fNewLine
       or (fmemInput.Lines[fintRowNumber][fintColNumber] = ' ')
```

```
       or (fmemInput.Lines[fintRowNumber][fintColNumber] = 'a')
       or (fmemInput.Lines[fintRowNumber][fintColNumber] = 'b')
       or (fmemInput.Lines[fintRowNumber][fintColNumber] = 'c')
       or (fmemInput.Lines[fintRowNumber][fintColNumber] = 'd')
       then goto O;
       if fTextEnd then begin flblHint.Caption := 'Ошибок в тексте не найдено';
Exit; end;
       if fmemInput.Lines[fintRowNumber][fintColNumber] = '1' then goto F2;
       if (fmemInput.Lines[fintRowNumber][fintColNumber] = '{') then goto A3
       else begin Error; Exit; end;
       end;
    F2 :
       begin
       fstrWord := fstrWord + fmemInput.Lines[fintRowNumber][fintColNumber];
       Transliterator;
       if fNewLine then
       begin
              flblHint.Caption := 'Обнаружена ошибка! Непредсказуемый  переход
на ' + IntToStr(fintRowNumber+1) + ' строчку';
              Exit;
       end;
       if fTextEnd then begin Error; Exit; end;
       if fmemInput.Lines[fintRowNumber][fintColNumber] = '0' then goto G2
       else begin Error; Exit; end;
       end;
    G2 :
       begin
       fstrWord := fstrWord + fmemInput.Lines[fintRowNumber][fintColNumber];
       Transliterator;
       if fNewLine then
       begin
              flblHint.Caption := 'Обнаружена ошибка! Непредсказуемый переход на
' + IntToStr(fintRowNumber+1) + ' строчку';
              Exit;
       end;
       if fTextEnd then begin Error; Exit; end;
       if fmemInput.Lines[fintRowNumber][fintColNumber] = '0' then goto EFIN2
       else begin Error; Exit; end;
       end;
      end;
{цифровая грамматика - конец}
{грамматика комментария - начало}
    '{' :
      begin
   A3 :
      Transliterator;
      if fTextEnd then begin Error; Exit; end;
      if fmemInput.Lines[fintRowNumber][fintColNumber] = '}' then
      begin Transliterator; goto O; end;
      if fmemInput.Lines[fintRowNumber][fintColNumber] <> '}' then
      goto A3;
      end;
{грамматика комментария - конец}
   else begin Error; Exit; end;
  end;

end;
{=====================================================================}
procedure TfrmMainForm.btnResetClick(Sender: TObject);
begin
  lblInfo.Caption := 'Добро пожаловать в Лексический Анализатор 1.0! Введите ваш
исходный текст в поле слева и нажмите "Выполнить".';
  memInput.Clear;
  btnReset.Enabled := false;
```

```
  btnRun.Caption := 'Выполнить';
end;
{=====================================================================}

procedure TfrmMainForm.btnRunClick(Sender: TObject);
begin
  lblInfo.Caption := 'Подождите, пожалуйста...';
  LexAn := TLexicalAnalyser.Create(frmMainForm.MemInput,frmMainForm.lblInfo);
  LexAn.Scaner;
  btnReset.Enabled := true;
  btnRun.Caption := 'Еще раз';
end;
{=====================================================================}
procedure TfrmMainForm.FormCreate(Sender: TObject);
begin
  btnReset.Enabled := false;
end;
{=====================================================================}

end.
```

ЛИТЕРАТУРА

1. Льюис Ф., Розенкранц Д., Стирнз Р. «Теоретические основы проектирования компиляторов». – М.: Мир, 1979.
2. Компаниец Р. И., Маньяков Е. В., Филатов Н. Е., «Системное программирование. Основы построения трансляторов». – СПб.: КОРОНА принт, 2000.
3. Мозговой М. В. «Классика программирования: алгоритмы, языки, автоматы, компиляторы. Практический подход». – СПб.: Наука и Техника, 2006.
4. Молчанов А. Ю. «Системное программное обеспечение: Учебник для вузов». – СПб.: Питер, 2003.
5. Синтез распознающего автомата. Методические указания к курсовой работе. – Новочеркасск: издание НПИ, 1987.