ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
Северо-Кавказский филиал
ордена Трудового Красного Знамени федерального государственного бюджетного образовательного учреждения высшего образования
"Московский технический университет связи и информатики"

Методические указания
для проведения лабораторной работы №7

по дисциплине

«СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ»

по теме

«Исследование работы учебного компилятора»

Направление подготовки:
09.03.01 Информатика и вычислительная техника

Профили **Программное обеспечение и интеллектуальные системы**
Вычислительные машины, комплексы, системы и сети

Ростов-на-Дону
2019

УДК 681.3.06 (076)
ББК 32.07

Чикалов А.Н. Системное программное обеспечение. Исследование работы учебного компилятора. Методические указания для проведения лабораторной работы №5. Ростов-на-Дону: Северо-Кавказский филиал МТУСИ, 2019.- 24 с.

В пособии изложены методические рекомендации и содержательные материалы для проведения занятий изучению системных ресурсов и алгоритмов работы файловой системы современных ОС, а также утилит для изучения, контроля и восстановления данных в файловой системе. Кроме того рассматриваются механизмы управления файловой системой программными средствами с помощью интерфейса прикладного программирования.

Пособие содержит необходимые справочные материалы.

Методические указания предназначены для студентов, обучающихся по направлению подготовки 09.03.01 Информатика и вычислительная техника, профиля Вычислительные машины, комплексы, системы и сети, Программное обеспечение и интеллектуальные системы.

Пособие предназначено для использования при изучении дисциплин Системное программное обеспечение, а также может быть использовано преподавателями и студентами при изучении родственных дисциплин и в процессе самостоятельной работы.

Учебное пособие обсуждено и одобрено на заседании кафедры ИВТ
Протокол №1 от 26.08.2019 г.

Рецензент Зав. кафедрой ИВТ д.т.н. профессор Соколов С.В.

Лабораторная работа 7. Исследование работы учебного компилятора.

1 Анализ предметной области

1.1 Компиляторы

Транслятор - это программа, которая переводит исходную программу в эквивалентную ей объектную программу. Если объектный язык представляет собой автокод или некоторый машинный язык, то транслятор называется компилятором.

Автокод очень близок к машинному языку; большинство команд автокода - точное символическое представление команд машины.

Ассемблер - это программа, которая переводит исходную программу, написанную на автокоде или на языке ассемблера (что, суть, одно и то же), в объектный (исполняемый) код.

Компиляторы пишутся как на автокоде, так и на языках высокого уровня. Кроме того, существуют и специальные языки конструирования компиляторов - компиляторы компиляторов.

Компилятор компиляторов (КК) – система, позволяющая генерировать компиляторы; на входе системы - множество грамматик, а на выходе, в идеальном случае, - программа. Иногда под КК понимают язык программирования, в котором исходная программа - это описание компилятора некоторого языка, а объектная программа - сам компилятор для этого языка. Исходная программа КК - это просто формализм, служащий для описания компиляторов, содержащий, явно или неявно, описание лексического и синтаксического анализаторов, генератора кодов и других частей создаваемого компилятора. Обычно в КК используется реализация схемы т.н. синтаксически управляемого перевода. Кроме того, некоторые из них представляют собой специальные языки высокого уров-

ня, на которых удобно описывать алгоритмы, используемые при создании компиляторов.

1.2 Логическая структура компилятора

На рисунке 1 представлена структурная схема компилятора.

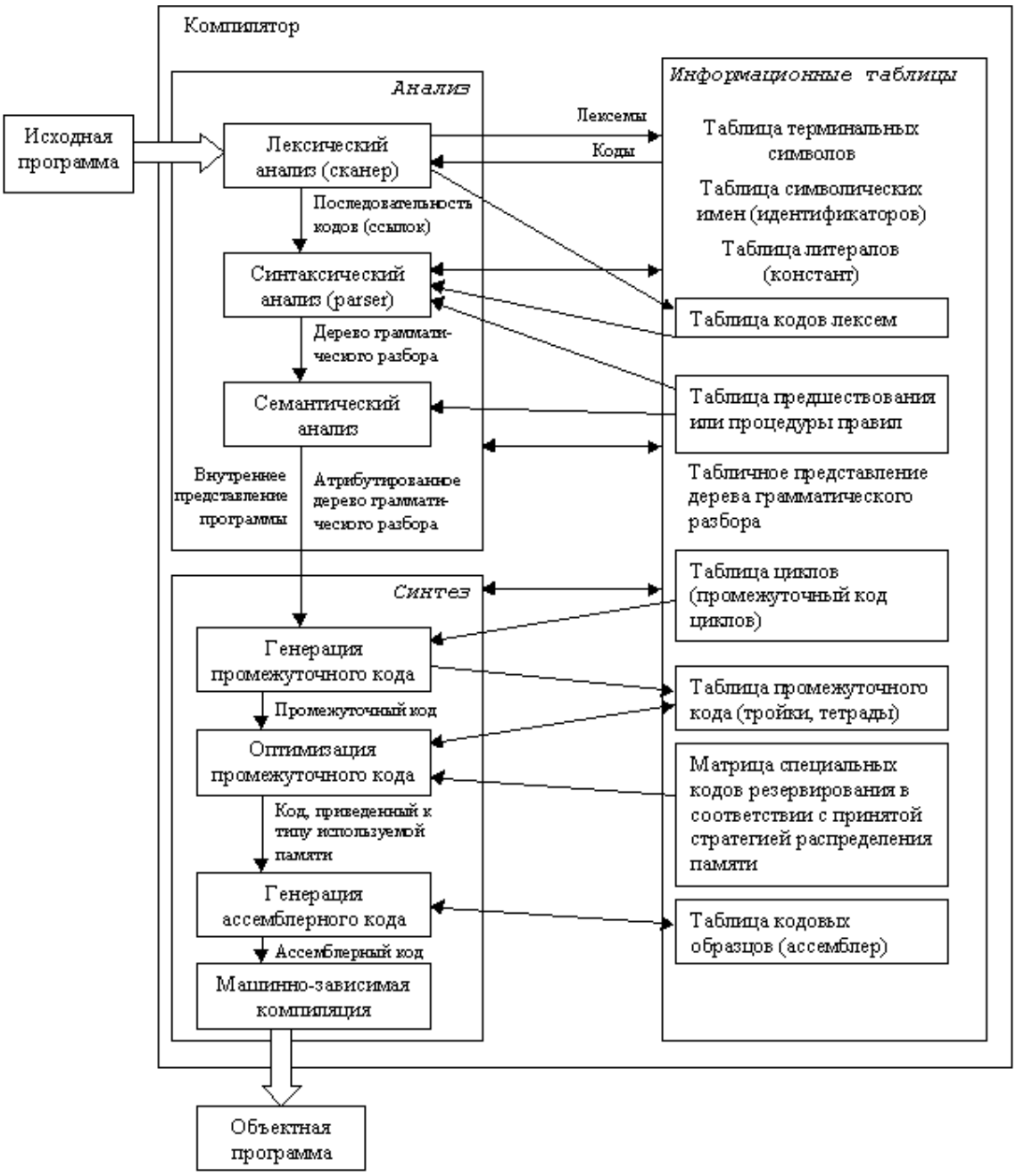

Рисунок 1 – Схема компилятора

Исходная программа – текст программы на языке высокого уровня (например Паскаль), который должен быть переведен в машинный код.

Информационные таблицы - самостоятельные структуры, заранее заполненные (таблица терминальных символов), а также заполняющиеся в ходе лексического анализа и дополняющиеся во время работы.

Лексический анализатор выполняет распознавание лексем языка и замену их соответствующими кодами. Под лексемами понимаются элементарные единицы, входящие в структуру предложения языка, такие как ключевые слова, константы, имена и т.п. Правильность задания структуры предложения языка на фазе лексического анализа не выполняется. Результатом является поток лексем (кодов – ссылок на таблицы), эквивалентный исходному тексту.

Синтаксический анализатор необходим для того, чтобы выяснить, удовлетворяют ли предложения, из которых состоит исходная программа, правилам грамматики этого языка.

Семантический анализ. На этом этапе осуществляется контроль типа и вида всех идентификаторов и других операндов.

Генерация промежуточного кода. Происходит преобразование исходной программы в промежуточную (например, польскую) форму записи.

Оптимизация промежуточного кода – выделение общих подвыражений и вычисление константных подвыражений.

Фаза оптимизации предназначена для уменьшения избыточности программы по затратам времени и памяти. В зависимости от критериев проектирования транслятора данная фаза обработки программы может исключаться из цикла обработки программы.

Распределение памяти. На этом этапе выделяются конкретные адреса пользователя под переменные, которые генерируются компилятором.

Генератор объектного (ассемблерного) кода – выполняет подстановку кодовых образцов на выходном языке, соответствующих промежуточным ко-

дам программы. Генератору кода могут не требоваться шаблоны, он весь может быть реализован в процедурном виде.

Машинно-зависимая компиляция. Зависит от того, какие используются регистры. Работа этой процедуры зависит от соглашений, принятых для исполняемой части программы. Например, выделяется базовый регистр для текущей активной записи в стеке.

На всех этих этапах происходит работа с различного рода таблицами. В частности, для каждого блока (если таковые существуют в языке) идентификаторы, описанные внутри, запоминаются вместе со своими атрибутами. Условно все эти этапы можно изобразить следующим образом:

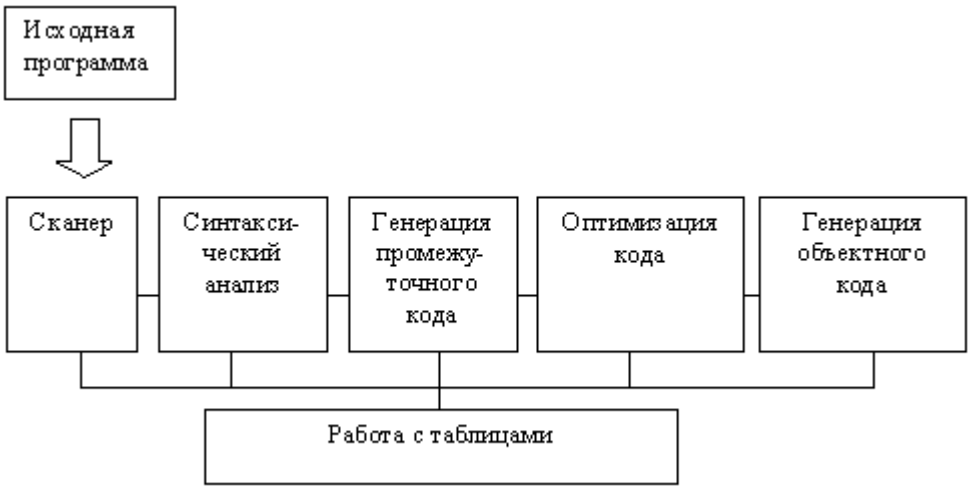

Рисунок 2 – Работа с таблицами

Очевидна зависимость структуры компилятора от структуры ЭВМ, точнее, от имеющейся производительности системы. Например, при малой памяти увеличивается количество проходов компиляции (т.н. многопроходные компиляторы), а при наличии памяти большого объема можно все этапы компиляции произвести за один проход (и тогда мы имеем дело с однопроходным компилятором).

Далее мы рассмотрим некоторые из этих составных частей процесса компиляции.

1.3 Лексический анализ. Сканер

Лексический анализатор представляет собой модуль разбора текста программы. Разбор происходит в зависимости от имеющихся у него в памяти терминальных символов и правилами определения типов данных. Каждый язык имеет свои ограничения по типам данных, допущения и особенности создания (строения) конструкций, применению операторов и т.п. При работе сканера используются три таблицы: таблица терминальных символов, таблица символических имен и таблица литералов – это заполняемые таблицы. Таблица терминальных символов хранит все ключевые слова и специальные символы используемые в языке, а также коды, соответствующие каждому символу. Таблица символических имен заполняется в процессе разбора текста программы и хранит в себе имена идентификаторов (символических имен). Таблица литералов также заполняется в процессе разбора программы и хранит в себе литералы: численные и строковые значения, с указанием типов данных и относительных адресов.

Распределение по таблицам происходит следующим образом.

Сканер в процессе анализа текста программы выделяет один из элементов текста и сравнивает с каждым терминальным символом. Если такой символ найден, то в выходной код передаются код таблицы и спецификатор (номер строки в таблице). В случае если этот элемент не является терминальным символом проверяется, является ли он идентификатором (первый символ обычно буква, остальные могут быть либо буквой, либо цифрой), если такой определен, то данный элемент заносится в таблицу символических имен, а к выходному

коду добавляется пара численных значений: номер таблицы и спецификатор найденного элемента. В случае если такой элемент в таблице уже имеется, то в выходной код заносится номер таблицы и его спецификатор. В таблицу литералов заносят численные, строковые и иные определенные языком значения. При этом распознается тип значения и тут же заполняется относительная таблица адресов.

Выходной код, сформированный сканером, передается на следующую стадию обработки – синтаксический анализ.

Таким образом, алгоритм работы простейшего сканера можно описать так:

- просматривается входной поток символов программы на исходном языке до обнаружения очередного символа, ограничивающего лексему;
- для выбранной части входного потока выполняется функция распознавания лексемы;
- при успешном распознавании информация о выделенной лексеме заносится в таблицу лексем, и алгоритм возвращается к первому этапу;
- при неуспешном распознавании выдается сообщение об ошибке, а дальнейшие действия зависят от реализации сканера - либо его выполнение прекращается, либо делается попытка распознать следующую лексему (идет возврат к первому этапу алгоритма).

Работа программы-сканера продолжается до тех пор, пока не будут просмотрены все символы программы на исходном языке из входного потока.

Таблица терминальных символов в простейшем случае может иметь следующий вид как показано в таблице 1.

Таблица 1 – Таблица терминальных символов

№ п.п.	Терминальный символ	Комментарий (обозначение)
1	PROGRAM	Заголовок программы
2	VAR	Описание переменных

3	BEGIN	Начало тела программы
4	END	Конец тела программы
5	INTEGER	Целый тип данных
6	FOR	Счетный оператор цикла (для)
7	TO	Ключевое слово счетного оператора цикла (до)
8	DO	Ключевое слово (выполнить)

Продолжение таблицы 1

№ п.п.	Терминальный символ	Комментарий (обозначение)
9	WHILE	Оператор цикла с предусловием (пока)
10	DIV	Деление целочисленное
11	MOD	Остаток от целочисленного деления
12	AND	Логическое И
13	OR	Логическое ИЛИ
14	:=	Присвоить значение

Сначала заполняется таблица лексем (терминальных символов), затем начинается считывание и обработка входного текста программы пользователя. При работе сканера происходит заполнение таблиц символических имен и литералов.

Данные таблицы могут выглядеть следующим образом:

Таблица 2 – Таблица символических имен

№ п.п.	Идентификатор	Тип	Размер, занимаемый в памяти, байт	Относительный адрес в памяти
1	I	INTEGER		
2	Y	REAL		
3	X1	REAL		
…				

Таблица 3 – Таблица литералов

№ п.п.	Литерал	Тип	Размер, занимаемый в памяти, байт	Относительный адрес в памяти
1	1	INTEGER	2	0
2	100	INTEGER	2	2
…				

Результатом работы сканера является последовательность кодов лексем. Каждый код лексемы обычно представляется кодом таблицы и спецификатором (порядковый номер в таблице, куда занесена лексема). Это позволяет избежать дополнительного поиска по таблицам на следующих этапах трансляции. Например в результате обработки сканером следующего предложения языка Паскаль

FOR I:=1 TO 100 DO Y:=X1

будет получена строка:

<1,06><2,1><1,14><3,1><1,07><3,2><1,08><2,2><1,14><2,3>,

где в угловых скобках пара чисел задает код таблицы и спецификатор. Можно оформить и в виде таблицы.

Таблица 4 – Таблица выходных символов

№ п.п.	1	2	3	4	5	6	7	8	9	10
Таблица	1	2	1	3	1	3	1	2	1	2
Строка	6	1	14	1	7	2	8	2	14	3

Функционально в сканере могут быть выделены следующие модули[4]:

1) выделение из входной строки очередного слова;
2) поиск в таблицах лексем и определение кода лексемы;
3) формирование и вывод выходной строки.

Для модуля выделения слова важна информация о том, какие символы могут быть признаками начала или конца слова. Например, в языке Паскаль ключевые слова отделяются от других элементов предложения пробелами. Сложнее обстоит дело с выделением идентификаторов и чисел, поскольку разделителем для них может служить любой другой символ, не входящий по определению в идентификатор или число.

При заполнении таблиц выполняется проверка на наличие в ней элемента, совпадающего с выделенным идентификатором или константой, и при совпадении занесение в таблицу не выполняется.

В задачи последнего модуля входит занесение в выходную строку кодов лексем.

В дополнение к своей основной функции, распознаванию лексем, сканер обычно также выполняет чтение строк исходной программы и, возможно, печать листинга исходной программы. Комментарии игнорируются сканером, за исключением того случая, когда они должны быть напечатаны и, таким образом, эффективно удаляются из исходной программы до начала процесса грамматического разбора.

Следующей стадией анализа является синтаксический разбор.

Лексический и синтаксический анализаторы взаимодействуют между собой. Существует два основных способа взаимодействия:

1) реализуется на основе прямого лексического анализа. От синтаксического анализатора поступает запрос «дать лексему» и указывается тип лексемы;
2) непрямой лексический анализ. Синтаксический анализатор выдает запрос «дать лексему», тип лексемы не указывается. Нет решающего блока, считаем, что работает группа параллельных автоматов.

1.4 Синтаксический и семантический анализ

Синтаксический анализ - это процесс, в котором исследуется цепочка лексем и устанавливается, удовлетворяет ли она структурным условиям, явно сформулированным в определении синтаксиса языка. Это – самая сложная часть компилятора.

Синтаксический анализатор расчленяет исходную программу на составные части, формирует ее внутреннее представление, заносит информацию в таблицу символов и другие таблицы. При этом производится полный синтаксический и, по возможности, семантический контроль программы. Фактически, это - синтаксически управляемая программа. При этом обычно стремятся отделить синтаксис от семантики насколько это возможно - когда синтаксический анализатор распознает конструкцию исходного языка, он вызывает семантическую процедуру, которая контролирует эту конструкцию, заносит информацию куда надо, проверяет на дублирование описания переменных, проверяет соответствие типов и т.п.

Процесс синтаксического анализа может рассматриваться как построение дерева грамматического разбора для транслируемых предложений. Грамматики могут использоваться как для порождения так и для распознавания предложений языка. Порождение начинается с начального понятия (или аксиомы грамматики). При распознавании с помощью грамматических правил порождается

предложение, которое затем сравнивается с входной строкой. При этом применение правил подстановки для порождения очередного символа предложения зависит от результатов сравнения предыдущих символов с соответствующими символами входной строки. Результат анализа исходного предложения в терминах грамматических конструкций удобно представлять в виде дерева. Такие деревья обычно называются *деревьями грамматического разбора* или синтаксическими деревьями. На рисунке 3 изображено дерево грамматического разбора для предложения READ (VALUE).

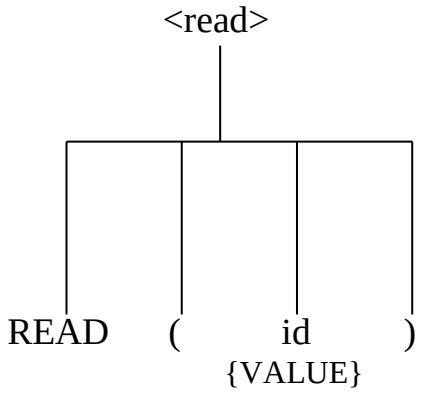

Рисунок 3 – Дерево грамматического разбора

Методы грамматического разбора разбиваются на два больших класса *восходящие* и *нисходящие* – в соответствии с порядком построения дерева грамматического разбора. Нисходящие (методы сверху-вниз) начинаются с аксиомы грамматики, с корня дерева и пытаются так его наращивать, чтобы последующие узлы дерева соответствовали синтаксису анализируемого выражения. Восходящие (методы снизу-вверх) начинают с элементов предложения (с "листьев") и отыскивают в грамматике какому понятию они соответствуют, т.е. определяют родительскую вершину для этих элементов, и т.д. пока не приходят к корню дерева (аксиоме грамматики). В современных компиляторах применяются как нисходящие, так и восходящие методы.

Достоинством восходящего метода является его несомненная простота, а также высокая скорость выполнения (не тратится время на поиск правила редукции).

Однако все эти достоинства напрочь меркнут перед главным недостатком данного метода. Дело в том, что здесь практически отсутствует какая бы то ни была диагностика (и тем более - локализация) ошибок. Во вторых, некоторые ошибки в исходном выражении не диагностируются вовсе. Например, выражения, в которых встречаются идущие подряд скобки “(” и “)”.

Поэтому при дальнейшем рассмотрении будет рассматриваться нисходящий разбор, как наиболее пригодный метод при ручном написании компилятора [4].

Кроме этого, алгоритмы синтаксического (грамматического) разбора (контроля) делят на синтаксически-ориентированные и синтаксически-неориентированные. Синтаксически-ориентированные алгоритмы являются универсальными для некоторого семейства языков и переход к распознаванию предложений другого языка выполняется путем смены грамматики, т.е. грамматика выполняет роль некоей "программы" распознавания предложений языка. Главным достоинством этого класса алгоритмов является их универсальность, а недостатком - наличие избыточности вследствие ориентации на семейство языков.

Синтаксически-неориентированные алгоритмы отличаются тем, что порядок действий в них определяется правилами грамматики данного конкретного языка. Достоинством этого класса алгоритмов является отсутствие избыточности, а недостатком - невозможность перенастройки на распознавание предложений другого языка.

В дальнейшем мы будем работать с синтаксически-неориентированными алгоритмами, т.к. будем работать лишь с одним языком – учебный язык на основе языка Паскаль.

1.5 Грамматики

Грамматика языка программирования является формальным описанием его *синтаксиса* или формы, в которой записаны отдельные предложения программы или вся программа. Грамматика не описывает *семантику* или значения различных предложений. Информация о семантике содержится в программах генерации объектного кода. В качестве иллюстрации разницы между синтаксисом и семантикой рассмотрим два предложения:

I:=J+K

и

I:=X+Y

где X и Y являются действительными переменными, а I, J, K — целыми переменными. Эти два предложения имеют одинаковый синтаксис. Оба являются операторами присваивания, в которых присваиваемое значение определяется выражением, состоящим из двух имен переменных, разделенных оператором сложения. Однако семантика этих двух предложений совершенно различна. Первое предложение говорит о том, что переменные в выражении должны быть сложены с использованием целых арифметических операций, а результат сложения должен быть присвоен переменной I. Второе предложение задает сложение с плавающей точкой, результат которого должен быть преобразован в целое число перед присваиванием. Очевидно, эти два предложения будут скомпилированы в различные последовательности машинных команд, хотя их грамматическое описание одинаково. Различия между ними проявятся на этапе генерации объектного кода.

На рисунке 4 показаны БНФ грамматики, используемые в дипломном проекте. Подчеркнутые волнистой линией элементы могут опускаться (не использоваться).

1. <prog> ::= PROGRAM <prog-name> VAR <dec-list> BEGIN <stmt-list> END.
2. <prog-name> ::= **id** ;
3. <dec-list> ::= <dec> { ; <dec> } ;
4. <dec> ::= <id-list> : <type>
5. <type> ::= INTEGER | REAL | STRING
6. <id-list> ::= **id** { , **id** }
7. <stmt-list> ::= <stmt> { ; <stmt> } ;
8. <stmt> ::= <assign> | <for> | <read> | <write> | <while> | <repeat> | <if>
9. <assign> ::= **id** := <exp>
10. <exp> ::= – <term> { + <term> | – <term> }
11. <term> ::= <factor> { * <factor> | DIV <factor> | / <factor> }
12. <factor> ::= **id** | **int** | **real** | <text-val> | (<exp>)
13. <read> ::= READ (<id-list>)
14. <write> ::= WRITE (<value> { , <value>})
15. <for> ::= FOR <index-exp> DO <body>
16. <index-exp> ::= **id** := <exp> TO|DOWNTO <exp>
17. <body> ::= <stmt> | BEGIN <stmt-list> END
18. <value> ::= <id-list> | <text-val>
19. <text-val> ::= ′ <text> ′
20. <text> ::= **string**
21. <if> ::= IF <сравнение> THEN <body> ELSE <body>
22. <сравнение> ::= <factor> <условие> <factor>
23. <условие> ::= > | < | = | >= | <= | <>
24. <while> ::= WHILE <сравнение> DO <body>
25. <repeat> ::= REPEAT <body> UNTIL <сравнение>

Рисунок 4 – Упрощенная грамматика языка Паскаль

Существует несколько различных форм записи грамматик, среди которых мы рассмотрим форму Бекуса—Наура (БНФ). БНФ не самое мощное из известных средств описания синтаксиса. Однако эта форма достаточно проста, широко используется и предоставляет достаточные для большинства приложений средства. На рис.4 изображена одна из возможных грамматик БНФ.

Грамматика БНФ состоит из множества *правил вывода,* каждое из которых определяет синтаксис некоторой конструкции языка программирования. Рассмотрим, например, правило 13 на рис. 4:

<read> ::= READ (<id-list>)

Это определение синтаксиса предложения READ языка Паскаль, обозначенное в грамматике как <read>. Символ ::= можно читать как «является по определению». С левой стороны от этого символа находится определяемая конструкция языка (в нашем случае— <read>), а с правой—описание синтаксиса этой конструкции. Строки символов, заключенные в угловые скобки < и >, называются *нетерминальными символами* (т. е. являются именами конструкций, определенными внутри грамматики). То, что не заключено в угловые скобки, называется *терминальными символами* грамматики (лексемами). В этом правиле вывода нетерминальными символами являются <read> и <id—list>. Терминальными символами являются лексемы READ, (,). Таким образом, это правило определяет, что конструкция <read> состоит из лексемы READ, за которой следует лексема (, за ней следует конструкция языка, называемая <id—list>, за которой следует лексема). Пробелы при описании грамматических правил не существенны и вставляются только для наглядности.

Для распознавания нетерминального символа <read> необходимо чтобы было определение для нетерминального символа <id-list>. Это определение дается правилом 6 на рис. 4:

<id-list> ::= **id** { , **id** }

Эта нотация, означает, что конструкция, заключенная в фигурные скобки, может быть либо опущена, либо повторяться один или более число раз. Таким образом, правило 6 определяет нетерминальный символ <id-list> как состоящий из единственной лексемы **id** или же из произвольного числа следующих друг за другом лексем **id,** разделенных запятой. В соответствии с этим новым определением процедура, соответствующая нетерминальному символу <id-list>, сначала ищет лексему **id,** а затем продолжает сканировать входной текст до тех пор, пока следующая пара лексем не совпадет с запятой и **id.** Такая запись устраняет проблему левой рекурсии.

1.6 Формирование промежуточного кода

Возможны различные формы внутреннего представления синтаксических конструкций исходной программы в компиляторе. Дерево грамматического разбора оказывается неудобным в работе при работе при генерации и оптимизации объектного кода. Поэтому перед оптимизацией и непосредственно генерацией объектного кода внутреннее представление программы преобразуется в одну из соответствующих форм записи.

Примерами таких форм записи являются:

- обратная польская запись операций;
- тетрады операций;
- триады операций;
- собственно команды ассемблера.

Обратная польская запись - это постфиксная запись операций. Преимуществом ее является то, что все операции записываются непосредственно в порядке их выполнения. Она чрезвычайно эффективна в тех случаях, когда для вычислений используется стек.

Тетрады представляют собой запись операций в форме из четырех составляющих:
<операция>(<операнд1>,<операнд2>,<результат>).

Тетрады используются редко, так как требуют больше памяти для своего представления, чем триады, не отражают взаимосвязи операций и, кроме того, плохо отображаются в команды ассемблера и машинные коды, так как в наборах команд большинства современных машин не встречаются операции с тремя операндами.

Триады представляют собой запись операций в форме из трех составляющих: <операция>(<операнд1>,<операнд2>), при этом один или оба операнда могут быть ссылками на другую триаду в том случае, если в качестве операнда данной триады выступает результат выполнения другой триады. Поэтому триады при записи последовательно номеруют для удобства указания ссылок одних триад на другие. Например, выражение A := B*C + D - B*10, записанное в виде триад будет иметь вид:

1) * (B, C)
2) + (^1, D)
3) * (B, 10)
4) - (^2, ^3)

5) := (A, ^4)

Здесь операции обозначены соответствующим знаком (при этом присвоение также является операцией), а знак ^ означает ссылку операнда одной триады на результат другой.

Команды ассемблера удобны тем, что при их использовании внутреннее представление программы полностью соответствует объектному коду и сложные преобразования не требуются. Однако использование команд ассемблера требует дополнительных структур для отображения их взаимосвязи. Кроме того, внутреннее представление программы получается зависимым от результирующего кода, а это значит, что при ориентации компилятора на другой результирующий код потребуется перестраивать как само внутреннее представление программы, так и методы его обработки в алгоритмах оптимизации (при использовании триад или тетрад этого не требуется).

Для построения внутреннего представления объектного кода (в дальнейшем - просто кода) по дереву вывода может использоваться простейшая рекурсивная процедура. Эта процедура прежде всего должна определить тип узла дерева - он соответствует типу операции, символ которой находится в листе дерева для текущего узла. Этот лист является средним листом узла дерева для бинарных операций и крайним левым листом - для унарных операций. После определения типа процедура строит код для узла дерева в соответствии с типом операции. Если все узлы следующего уровня для текущего узла есть листья дерева, то в код включаются операнды, соответствующие этим листьям, и получившийся код становится результатом выполнения процедуры. Иначе процедура должна рекурсивно вызвать сама себя для генерации кода нижележащих узлов дерева и результат выполнения включить в свой порожденный код.

Поэтому для построения внутреннего представления объектного кода по дереву вывода в первую очередь необходимо разработать формы представления объектного кода для четырех случаев, соответствующих видам текущего узла дерева вывода:

оба нижележащих узла дерева - листья (терминальные символы грамматики);

только левый нижележащий узел является листом дерева;

только правый нижележащий узел является листом дерева:

оба нижележащих узла не являются листьями дерева.

Метод четверок

Каждая четверка записывается в виде:

операция, op1, op2, результат,

где операция - это выполняемая объектным кодом функция

op1, op2 - операнды этой операции

Например,

(-a+b)*(c+d)

будет соответствовать такой последовательности четверок

- a, T1
+ T1, b T2
+ c, d T3
* T2, T3, T4

Из сформированных четверок нетрудно сгенерировать машинный код.

1.7 Обоснование создания учебного комплекса

Создание учебного комплекса обосновано сложностью предметной области создания компиляторов и отсутствием подобного наглядного материала, позволяющего поэтапно отследить процессы, происходящие в компиляторе (работа анализаторов, заполнение таблиц).

Из описанного выше материала видно, что компиляторы имеют очень много вариантов построения, это определяет дополнительную сложность при выборе оптимальной структуры компилятора. Процесс компиляции довольно сложен и труден для восприятия в целом. Он содержит в себе множество понятий, подходов обработки, методов разбора, таблиц, формальные грамматики. Поэтому целесообразно разбиение комплекса на ряд лабораторных работ, в которых будут описаны практические методики изучения различных этапов компиляции на основе специальных обучающих программ.

1.8 Обзор существующих разработок

На текущий момент существует множество разработок, связанных с созданием компиляторов. В основном это методические, учебные пособия по проектированию компиляторов, где рассматриваются основные принципы анализа текста программы и синтеза объектного кода. В основном в этих материалах приводятся примеры как написать оптимальную программу компиляции по скорости, используя стеки, таблицы предшествования и т.п. структуры, не позволяющие или затрудняющие передавать данные между этапами как требуется в учебном комплексе.

Работа с методическим (теоретическим) материалом подразумевает выполнение задания с дальнейшей проверкой преподавателем, что затрачивает его время и силы. Программная реализация дает возможность самому проконтролировать себя на правильность проведенного анализа или корректность синтезированного кода.

Руководство по написанию компиляторов Креншоу [5] позволяет создать свой компилятор, но его особенностью является прямой перевод считанного текста в выходной код, что не дает наглядности внутреннего представления компилятора. Этот компилятор является однопроходным, он не хранит и создает в памяти таблиц, вся обработка описывается в процедурах.
Существующие компиляторы не дают наглядное представление внутренних структур, а обычно имеют один исполняемый файл, реализующий перевод текста программы в объектный или исполняемый файл. Ставка обычно делается либо на скорость компиляции, либо на минимальный размер генерируемого файла.

Компиляторы компиляторов в основной своей массе создают лишь лексические анализаторы, оставляя дорабатывать синтаксическую часть самому программисту. При обзоре не было найдено российских разработок компиляторов компиляторов, что так же говорит о том, что при работе с существующими разработками добавится еще одна проблема – проблема языка.

1.9 Обоснование разработки

Разработка является учебным комплексом, включающим в себя лабораторные работы по изучению процесса компиляции. Это программа, позволяющая без руководителя выполнять и проверять выданные задания (проверять выполненные задания). Теоретический материал для проведения лабораторной работы сформирован в методической (описательной) части.

Особенность данной разработки в том, что спроектированный компилятор построен поэтапно (модульно). Результат работы каждого из этапов может быть зафиксирован (сохранен) в виде файла с определенной структурой. Это файл с промежуточным кодом, получаемый от сканера, файл с формируемой таблицей переходов (хранит структуру дерева), файл с промежуточным кодом (тетрады), ассемблерный код. На каждом из этапов имеется возможность генерации файла отчета со структурами, с описанием и указанием ошибок, а также дополнительной служебной информацией.

Данный комплекс служит для ознакомления с принципами компиляции, получения практических навыков лексического анализа и грамматического разбора (синтаксический анализ), формирования промежуточного кода. Комплекс построен таким образом, чтобы по возможности охватить все этапы компиляции, наглядно представляя формируемые таблицы.

При проектировании (разработке, планировании) комплекса ставка делалась на наглядность происходящих процессов и доступность для понимания правил формирования и заполнения множества таблиц. При работе с программным продуктом не показана работа со стеком, т.к. вся реализация, весь анализ происходит только с таблицами и только в таблицах, исключение составляет разве что входной текст программы и выходной код.

Учебное пособие состоит из:

- вводной части (теоретические сведения):
  1) описание компиляторов, их суть, назначение;
  2) лексический анализатор (сканер);

3) синтаксический анализатор, дерево грамматического разбора;

4) получение промежуточного кода;

- практической (работа с программами):

1) обзор компиляторов (Паскаль, C, Delphi);

2) работа с программой LexAn;

3) работа с программой SinAn;

4) работа с программой SinAn;

- проверка полученных знаний с помощью контрольных вопросов и заданий.

Учебный комплекс служит для облегчения работы преподавателя, возможности самостоятельно изучения материала, получения практических навыков по изучаемой дисциплине, возможности более наглядного представления информации и т.п.

Учебный комплекс включает в себя несколько взаимосвязанных лабораторных работ, охватывающих всю предметную область или основную ее часть, например обучение процессу компиляции. При этом при выполнении каждой лабораторной работы происходит поэтапное изучение предметной области.

Лабораторные работы обычно включает в себя:

- теоретические сведения;
- порядок выполнения работы;
- контрольные вопросы и задания.

Теоретические сведения дают представление об изучаемой области, ознакомление с ее основными принципами, структурами и характерными особенностями. При этом часто производится разбор какого-либо наглядного примера.

Для проведения лабораторных работ могут использоваться различные технические средства. Это могут быть различного рода стенды, имитирующие работу реальных устройств, сами устройства, выступающие в роли исследуемого объекта, компьютер, с набором необходимых для работы программ, а также другие устройства и оборудование, подходящие для этой цели.

Использование в лабораторных работах оборудования позволяет получать дополнительные практические навыки, когда студент может влиять на работу исследуемого объекта, изменяя различные входные и управляющие пара-

метры. При этом учащийся лучше понимает всю картину происходящего, исследуемые процессы.

Во время выполнения лабораторных работ часто приходится снимать показания с приборов, получать различные данные от датчиков, программ и т.п., заносить их в таблицы и обрабатывать соответствующим образом. При этом производятся расчеты, связанные с работой, оформляется отчет, который и сдается преподавателю на проверку.

Контрольные вопросы формируют исходя из цели проведения лабораторной работы и того, что должен вынести обучающийся в результате ее выполнения: определения, термины, понятия, связанные с изучаемым объектом, принципы его работы, строение.

2 Создание учебной разработки

2.1 Краткое описание учебного компилятора

Учебный компилятор состоит из четырех отдельных модулей, это:

1) лексический анализатор (сканер) LEXAN;
2) синтаксический анализатор (парсер) SYNAN;
3) генератор промежуточного кода PROMKOD;
4) генератор ассемблерного кода ASMKOD.

На данном этапе реализованы первые два. Эти модули (этапы) взаимодействуют между собой с помощью промежуточных файлов.

Среда LEXAN генерирует файл с расширением LEX, в котором хранятся таблицы, полученные в результате разбора текста программы: таблица выбранных терминальных символов, таблица символических имен, таблица лексем и таблица выходных кодов лексем, которая и представляет собой программу в виде ссылок на три предыдущие таблицы. Данный файл является входным на этапе синтаксического анализа.

Среда SINAN генерирует файл с расширением SYN, хранящий в себе формируемую таблицу переходов, представляющую собой грамматическое дерево в табличном виде. В этом же файле хранятся таблицы выбранных терминальных символов, символических имен и лексем. Данный файл является входным на этапе генерации промежуточного кода.

Среда PROMKOD генерирует файл PRK, хранящий в себе упрощенное дерево грамматического разбора, представленное в виде таблицы триад.

Среда ASMKOD генерирует файл ASK, представляющий собой программу на ассемблере.

В результате проведенного анализа была выбрана многопроходная схема просмотра компилятора. На каждом этапе (лексический анализ, синтаксический анализ, формирование промежуточного кода, формирование ассемблерного кода) происходит новый просмотр (проход) по программе, представленной в различном виде. На первом этапе (сканер) – в виде текста программы, на втором (парсер) – в виде кодов лексем, на третьем – дерево грамматического разбора, на четвертом – таблица промежуточного кода. Это сделано для поэтапного обучения процессу компиляции и возможности работы с внутренним представлением программы.

Все данные, кроме входного текста программы помещаются в таблицы. Это сделано для того, чтобы не использовать стек и все данные представлять по возможности в одном месте.

При выборе языка высокого уровня, в качестве входного языка для анализа был принят учебный язык, основанный на упрощенном варианте языка Паскаль. Язык Паскаль является довольно распространенным, довольно понятным и простым для восприятия, к тому же его структуры довольно удобны для разбора. Описание учебного языка приведено ниже.

2.2 Описание учебного языка

Учебный язык построен на основе языка Паскаль.

Алфавит *учебном* языка включает буквы, цифры, специальные символы и зарегистрированные слова.

Буквы – это буквы латинского алфавита от a до я, от A до Я, от a до z и от A до Z. В данном языке нет различия между прописными и строчными буквами алфавита, если только они не входят в символьные и строковые выражения.

Цифры – арабские цифры от 0 до 9.

Специальные знаки *учебного* языка – это символы:

+ - * / = , . : ; < > { } [ ] ( )

К специальным знакам также относятся следующие пары символов:

<> <= >= :=

в программе эти символы нельзя разделять пробелами, если они используются как знаки операций отношения.

Особое место в алфавите языка занимают пробелы. Эти символы рассматриваются как ограничители идентификаторов, констант, чисел, зарезервированных слов. Несколько следующих друг за другом пробелов считаются одним пробелом.

В учебном языке имеются следующие зарезервированные слова:

and	function	then
begin	if	to
div	integer	until
do	procedure	var
downto	program	while
else	real	write
end	repeat	read
for	string	

Их можно изменять при построении компилятора в соответствующей программной среде LEXAN.

Идентификаторы – имена переменных, процедур, функций, программ. Длина идентификатора ограничена 255 символами. Идентификатор всегда начинается буквой или знаком подчеркивания, за которым могут следовать буквы, цифры и знак подчеркивания. Пробелы и специальные символы не могут входить в идентификатор.

Константы.

Последовательность, состоящая из одной или более цифр 0, 1, … , 9, является целой (INTEGER) константой. Данный тип занимает в памяти 2 байта. Последовательность цифр, разделенных точкой, является вещественной (REAL) константой, данный тип занимает в памяти 4 байта. Последовательность любых символов (кроме знака одинарных кавычек), за-

ключенных в одинарные кавычки, является строковой (STRING) константой, длина данного типа варьируется от 1 до 255 байт, в зависимости от числа символов в последовательности.

Выражения.

Операции в выражении выполняются слева направо; как обычно, учитывается наличие скобок и приоритеты операторов. Приоритеты операторов приведены в таблице 5 (оператор в первой строке имеет наивысший приоритет):

Таблица 5 – Таблица приоритетов

– (унарный)
* / div
+ – (бинарный)
= <> < > <= >=

Ключевые слова, идентификаторы, лексемы отделяются друг от друга пробелами, от специальных символов разделение не обязательно.

Возможные для использования символы:

буквы: а..я, А..Я, a..z, A..Z;

символ, разрешенный при написании имен: _

элементы разделения: , ; : пробел

разделитель целой и дробной частей в вещественных числах: .

выделение текста: ′

знаки операторов: + - * /

комментарии: { }

расстановка приоритетов: ( )

знаки сравнения: > < = >= <= <>

признак окончания программы: .

2.3 Лексический анализатор LEXAN

Цель создания программы LEXAN состоит в том, чтобы научить студента производить разбор текста программы на составляющие ее лексемы в соответствии с заданной БНФ, при этом правильно заполнив таблицы выбранных терминальных символов, символических имен, литералов и выходных кодов лексем.

Данная среда позволяет сравнить данные, внесенные студентом с данными, полученными программой и сгенерировать сообщения об ошибках, на основе которых студент будет иметь возможность внести соответствующие исправления.

При выполнении дипломного проекта был проведен анализ способов построения лексического анализатора. За основу был принят прямой синтаксический анализатор, так как считывает лексему, находящуюся справа от указателя и лишь потом определяет тип лексемы [3]. Кроме того, отчасти используется непрямой анализ при отделении специальных символов от идентификаторов, ключевых слов и литералов, когда разделительный пробел не обязателен.

Лексический анализатор позволяет работать со следующими таблицами:

1) таблица выбранных терминальных символов (формируется из таблицы терминальных символов);
2) таблица символических имен (идентификаторов);
3) таблица литералов (констант);
4) таблица выходных кодов лексем.

Далее описываются структуры таблиц.

2.3.1 Таблица терминальных символов

Внутри программы хранится таблица терминальных символов. Она хранит в себе все терминальные символы, которые могут использоваться в учебном языке (ключевые слова и специальные символы). Они имеют свои названия, описание и каждому ключевому слову соответствует свой уникальный код, по которому происходит идентификация элемента на следующих стадиях компиляции. На данном этапе происходит работа с таблицей выбранных терминальных символов, пример которой показан в таблице 6.

Таблица 6 – Таблица выбранных терминальных символов

№ стр.	Терминальный символ	Комментарий	Код
1	PROGRAM	Объявление переменных	1

Таблица выбранных терминальных символов содержит следующие поля:

№ стр – номер строки в таблице выбранных терминальных символов;

Терминальный символ – название терминального символа;

Комментарий – описание терминального символа;

Код – код терминального символа, определенный в таблице терминальных символов.

Данная таблица формируется из таблицы терминальных символов, определенной внутри программы (описана в приложении А) путем выбора необходимых терминальных символов в соответствующем окне программы. Она служит (необходима) для проверки, является ли полученная лексема терминальным символом или идентификатором, т.е. производится сравнение со всеми

терминальными символами таблицы. Если лексема найдена в таблице, то в таблицу выходных кодов лексем заносится номер таблицы (в программе №1) и код терминального символа.

Некоторые терминальные символы можно изменять – это ключевые слова. Изменение возможно в момент заполнения таблицы выбранных терминальных символов.

2.3.2 Таблица символических имен

Для хранения значений идентификаторов служит таблица символических имен, пример которой приведен в таблице 7.

Таблица 7 – Таблица символических имен

Специф	Идентификатор	Тип	Размер в памяти	Относительный адрес в памяти
1	a			

Таблица символических имен содержит следующие поля:

Специф – спецификатор (номер строки) определяет положение идентификатора в таблице;

Идентификатор – имя идентификатора, найденного в тексте программы;

Тип – тип распознанного идентификатора (заполняется в программе LEXAN), поле остается не заполненным;

Размер в памяти – размер идентификатора, занимаемый в памяти, определяется в зависимости от типа (заполняется в программе LEXAN), поле остается не заполненным;

Относительный адрес в памяти – адрес относительно начала объявления переменных, формируется в зависимости от размера памяти предыдущих идентификаторов (заполняется в программе LEXAN), поле остается не заполненным.

Таблица служит для хранения идентификаторов, найденных в тексте программы. После внесения идентификатора или обнаружения уже имеющегося в таблице, в таблицу выходных кодов лексем заносится номер таблицы (№2) и спецификатор найденного элемента.

2.3.3 Таблица литералов

Для хранения значений констант используется таблица литералов, пример ее заполнения показан в таблице 8.

Таблица 8 – Таблица литералов

Специф	Литерал	Тип	Размер в памяти
1	10	INTEGER	2

Таблица содержит следующие поля:

Специф – спецификатор, определяет положение идентификатора в таблице;

Литерал – значение литерала, найденного в тексте программы;

Тип – тип распознанного литерала;

Размер в памяти – размер литерала, занимаемый в памяти, определяется в зависимости от типа;

Относительный адрес в памяти – адрес относительно начала объявления переменных, формируется в зависимости от размера памяти занимаемой литералами и идентификаторами.

Таблица служит для хранения литералов, найденных в тексте программы. После внесения литерала в таблицу, в таблицу выходных кодов лексем заносится номер таблицы (№3) и спецификатор найденного элемента.

2.3.4 Работа сканера

Работа сканера LEXAN происходит следующим образом. Студент в соответствующее поле пишет (или загружает из файла через меню) текст программы. Далее выбираются терминальные символы, необходимые для разбора текста программы и на основе правил разбора заполняются таблицы символических имен, литералов и выходных кодов лексем. После этого производится проверка правильности заполнения. При этом программа производит анализ текста и заполняет свои внутренние соответствующие таблицы и сравнивает с данными, полученными от студента и, при наличии ошибки, генерирует сообщения в поле сообщений. При необходимости можно получить листинг. Также

можно сохранить результаты в файл для передачи данных на следующий этап – синтаксический анализ.

2.3.5 Структура листинга

Листинг включает в себя текст программы, все таблицы с заполненными студентом данными и все сообщения об ошибках.

2.3.6 Структура выходного файла

Выходной файл хранит в себе все 4 таблицы, построчно храня каждую ячейку. Это позволяет не ограничивать длину идентификаторов и ключевых слов. Вначале файла также построчно указываются размеры таблиц, сначала выбранных терминальных символов (число столбцов, число строк), затем символических имен, литералов и, наконец, выходных кодов лексем (только число столбцов). Структура промежуточного файла показана в таблице 9.

Таблица 9 – Пример промежуточного файла

№ строки	Содержи-	Описание содержимого

	мое	
1	5	4 столбца + 1 (четвертый) зарезервирован
2	7	1 строка – заголовок таблицы, последующие 6 – строки с данными
3	5	5 столбцов
4	4	1 строка – заголовок таблицы, последующие 3 – строки с данными
5	5	5 столбцов
6	3	1 строка – заголовок таблицы, последующие 2 – строки с данными
7	16	1 столбец – описания, остальные 15 – с данными (в таблице три строки)
8		данные из таблицы 1 по ячейкам, следуют слева направо (построчно), сверху вниз.
…		
7+5*7=42		
43		данные из таблицы 2 по ячейкам, следуют слева направо (построчно), сверху вниз.
…		
42+5*4=62		
63		данные из таблицы 3 по ячейкам, следуют слева

…		направо (построчно), сверху вниз.
62+5*3=77		
78		данные из таблицы 4 по ячейкам, следуют сверху вниз (по столбцам), слева направо.
…		
77+16*3=125		

В качестве примера приводится пример разбора задания, описанного в приложении А.

2.3.7 Примерное задание для студента

Дана некоторая грамматика языка:

1. <prog> ::= PROGRAM <prog-name> VAR <dec-list> BEGIN <stmt-list> END.
2. <prog-name> ::= **id**
3. <dec-list> ::= <dec> | <dec-list> ; <dec>
4. <dec> ::= <id-list> : <type>
5. <type> ::= INTEGER
6. <id-list> ::= **id** | <id-list> , **id**
7. <stmt-list> ::= <stmt> | <stmt-list> ; <stmt>
8. <stmt> ::= <assign> | <for>
9. <assign> ::= **id** := <exp>
10. <exp> ::= <term> | <exp> + <term> | <exp> - <term>
11. <term> ::= **id** | **int** | (<exp>)
12. <for> ::= FOR <index-exp> DO <body>
13. <index-exp> ::= **id** := <exp> TO <exp>

14. <body> ::= <stmt> | BEGIN <begin-list> END

Используя программу LEXAN произвести следующие действия:

1. Заполнить таблицу терминальных символов (таблица 1);
2. Написать исходный текст на учебном языке с использованием заданной грамматики;
3. Заполнить таблицы: – символьных имен (таблица 2);
   – литералов (таблица 3);
   – лексического анализа (выходных символов);
4. Проверить правильность заполнения таблиц встроенным анализатором;
5. При наличии ошибок исправить имеющиеся, и повторно обработать программой LEXAN;
6. Получить листинг полученных результатов;
7. Сохранить результат в файл.

Пример выполнения приведен в приложении А.

2.3.8 Описание работы лексического анализатора

После того как студент написал программу или загрузил ее из файла, также заполнил все таблицы и запустил программу на проверку, программа начинает выполнять следующее.
Программа производит чтение первого символа, далее производятся проверки.

- Если считанный символ является буквой или знаком подчеркивания «_», если да, то это либо ключевое слово, либо идентификатор. Далее считывается следующий символ (литера) и производится его проверка, входит ли этот символ во множество букв русского и латинского алфавитов, цифр, является ли он символом подчеркивания, если да, то полученный символ добавляется к строковой переменной, формирующей лексему. Дальнейшее считывание и обработка происходит до тех пор, пока не встретится какой либо другой символ.
- Если считанный символ является цифрой, то далее происходит проверка, является ли следующий символ цифрой или точкой. Если полученная литера состоит из од-

них цифр, то полученное число целого (INTEGER) типа, если в литере есть точка, то число вещественного (REAL) типа.

- Если считанный символ – одинарная кавычка, то текст, следующий за ней до следующей одинарной кавычки, будет являться строковой константой, а знаки кавычек будут определены как специальные символы.
- Если считанный символ является знаком «{», то сам знак и следующие за ним символы до знака «}» включительно игнорируются, так как являются комментарием.
- Если считанный символ является специальным символом, происходит проверка, является ли данный символ сдвоенным и проверяется второй символ. Если второй символ не образует пару или первый из двух найденных является одинарным, то происходит обработка данного терминального символа, поиск его кода.

После распознавания лексемы происходит заполнение таблиц, соответствующих типу лексемы. Если предполагается, что полученная лексема является терминальным символом, то происходит перебор всех значений таблицы терминальных символов. В случае, когда лексема найдена, необходимо получить ее код и заполнить соответствующим образом таблицу выходных кодов лексем. В случае, когда лексема не найдена предполагается, что лексема является идентификатором. Происходит поиск по таблице символьных имен. В случае, когда в таблице такая лексема уже имеется, происходит заполнение таблицы выходных кодов лексем, иначе лексема включается в таблицу и также заполняется таблица кодов лексем.

При обнаружении литерала, найденная лексема заносится в таблицу, в соответствующем поле заносится ее тип, далее указывается его размер в байтах. Затем заполняется таблица выходных кодов лексем.

В порядке распознавания лексем происходит заполнение таблицы выходных кодов лексем. Если распознанная лексема является терминальным символом, то в ячейку, соответствующую номеру таблицы, заносится номер 1, если является идентификатором – номер 2, если литералом – 3. Спецификатор («код» для терминального символа) заносится в поле «Строка».

Далее происходит пошаговое сравнение значений, полученных программой, со значениями внесенными студентом. Сравнение происходит по таблице выходных кодов лексем. При каждом несоответствии генерируется сообщение в окне сообщений, что в такой-то позиции не верно заполнено значение номера таблицы, кода элемента, спецификатора.

Имеется возможность получения листинга в отдельный файл с расширением LOG.

Кроме того, необходимо сохранить файл для работы на следующем этапе синтаксического анализа.

2.4 Синтаксический анализатор SinAn

Цель создания программы SINAN состоит в том, чтобы научить студента проверять правильность грамматики программы с помощью синтаксических деревьев (деревьев грамматического разбора).

Программа SINAN сама производит разбор программы, строит синтаксическое дерево и проверяет введенные пользователем данные на корректность, сообщая обо всех найденных ошибках и несоответствиях.

2.4.1 Таблица переходов

Существует два пути анализа: восходящий и нисходящий, данный проект реализован с помощью нисходящего, он называется рекурсивный спуск. В проекте грамматический разбор реализован с помощью правил БНФ грамматики, заданных в таблице переходов.

В таблице переходов с помощью специальных кодов реализованы ссылки, переходы, обозначения терминальных символов, идентификаторов и литералов, см. таблицу 10. Нетерминальные символы представляют собой ссылки на конструкции, терминальные – указатели на код элемента соответствующей таблицы, идентификаторы и литералы представляют собой соответствующие обозначения. Для решения проблемы выбора одного из нескольких вариантов введен элемент ИЛИ, позволяющий реализовать все возможные варианты ветвления. Для реализации стека в каждой строке предусмотрена ячейка возврата, в

которой указывается адрес, куда следует перейти после отработки соответствующей конструкции.

На основе данной таблицы производится анализ кодов лексем и создается новая формируемая таблица переходов, по которой в дальнейшем строится синтаксическое дерево.

Таблица переходов полностью основана на БНФ грамматике, показанной на рис. 4. Эта таблица предназначена для реализации синтаксического разбора с помощью метода рекурсивного спуска. С помощью нее можно определить законченность выражений, отследить грамматику учебного языка. Она служит основной базой при написании программы, хотя ее можно использовать и для построения формируемой таблицы переходов вручную.

На основе этой таблицы формируется другая (которую при необходимости легко можно преобразовать в дерево грамматического разбора), конечная таблица представляет собой программу, разобранную по грамматикам (на грамматики), представленную переходами (ссылками) и адресами таблиц и спецификаторов (№-в строк) на хранящиеся в них данные.

Работа с данной таблицей не оптимальна по скорости, т.к. при работе не используется стек, зато данное представление более наглядно.

Таблица 10 – Таблица переходов

		1	2	3	4	5	6	7	8	9
1	<prog>		PROGRAM| 1,4 $1 |@1,4	<prog-name> ~2	VAR| 1,6 $2 | @1,6	<dec-list> ~3	BEGIN $3	<stmt-list> ~7	END $4	. $30
2	<prog-name>		 +id	; $27						
3	<dec-list>		<dec> ~4	; $27	<dec>| ~4 | @3,1	; $27	 @3,4			
4	<dec>		<id-list> ~6	: $31	<type> ~5					
5	<type>		INTEGER| REAL | STRING $5 | $6 | $7							
6	<+id-list>		 +id	, | $29| @6,1	 +id	 @6,3				
7	<stmt-list>		<stmt> | ~8 | @7,1	; | $27| @7,1	 @7,2					
8	<stmt>		<assign>|<for>|<read>|<write>|<for>|<while>|<repeat>| <if> ~9 | ~15 | ~13 | ~14 | ~15 | ~24 | ~25 | ~21							
9	<assign>		 +id	:= $28	<exp> ~10					
10	<exp>		- | + | $33 | $32 | @10,3	<term> ~11	+ | - | $32| $33| @10,1	<term> ~11	 @10,4			
11	<term>		<factor> ~12	* | DIV | / | $34| $17 | $37|11,1	<factor> ~12	 11,3				
12	<factor>		 +id | ^int | ^real | ^string |@12,4		($35| @12,1	<exp> ~10	) $36			
13	<read>		READ $19	($35	<id-list> ~6	) $36				
14	<write>		WRITE $18	($35	<VALUE> ~18	, | 14,9 $29| @14,9	<VALUE> ~18	 @14,5		) $36
15	<for>		FOR $8	<index-exp> ~16	DO $10	<body> ~17				
16	<index-exp>		 +id	:= $28	<exp> ~10	TO|DOWNTO $9 | $20	<exp> ~10			
17	<body>		<stmt>| 17,4 ~8 | @17,4		BEGIN $3	<stmt-list>	END $4	; | $27| @17,1		
18	<value>		<id-list>| <text-val> ~6 | ~19							

Продолжение таблицы 10

19	<text-val>		′ $38	<text> ~20	′ $38					
20	<text>		 ^string							
21	<if>		IF $14	<сравнение> ~22	THEN $15	<body> ~17	ELSE | $16 | @21,1	<body> ~17		
22	<сравнение>		<factor> ~12	<условие> ~23	<factor> ~12					
23	<условие>		< | > | = | >= | <=| <> $39|$40|$41|$42|$43|$44							
24	<while>		WHILE ~13	<сравнение> ~22	DO $10	<body> ~17				
25	<repeat>		REPEAT $11	<body> ~17	UNTIL $12	<сравнение> ~22				

2.4.2 Правила работы с таблицей переходов

Ячейкой возврата является первая ячейка каждой строки, ее описание: X,1, где X – строка, 1 – столбец.

~x – переход на строку с номером x, столбец №2, формирование адреса возврата в первой ячейке, если переход был осуществлен от одного из параметров условий ИЛИ, то формирование адреса возврата и адреса перехода при ошибке (отрицательном результате).

| – элемент ИЛИ, предпочтение отдается более левому элементу. Перебором сравнивается каждый элемент, и если не один не подошел, то ошибка.

$x – в таблице переходов – номер строки в таблице терминальных символов

$x,y – в формируемой таблице переходов – одна из трех таблиц (x=1 – терминальных символов, x=2 –идентификаторов, x=3 – литер);

+id – таблица идентификаторов (№2) в таблице переходов.

Элементы, начинающиеся со знака ^ (int, real, string) в таблице переходов являются элементами таблицы литералов (№3).

@x,y,z – переход в строку x, столбец y, параметр z

@x,y,z%a,b,c – переход в строку x, столбец y, параметр z (при наличии элемента ИЛИ), при наличии ошибки в a,b,c (указывается только в ячейках возврата, формируется только при присутствии элемента ИЛИ)

Переход по ошибке (значение после знака % в ячейке возврата) действует только для ячеек столбца №2.

Алгоритмы:

Используются таблицы Table_Perehod[i,j], Table_Perehod1[i1,j1]

Table_Perehod –основная таблица перехода

Table_Perehod1 – формируемая таблица перехода

[i,j] – адреса ячеек внутри Table_Perehod, i – столбцы, j – строки

[i1,j1] – адреса ячеек внутри Table_Perehod1, i1 – столбцы, j1 – строки

count_vs – счетчик перемещения, показывающий текущее положение в таблице выходных символов (Tabl_vs);

pos – номер позиции в ячейке (при наличии элемента ИЛИ).

Таблица №1 – таблица терминальных символов.

Таблица №2 – таблица символических имен (идентификаторов).

Таблица №3 – таблица литералов.

Просмотр осуществляется слева направо, и по переходам. Каждый раз происходит сравнение с текущим элементом из таблицы выходных символов.

При положительном результате сравнения (терминальных символов, идентификаторов, литер), осуществляется переход на более правую ячейку.

При отрицательном результате (err=1), обработка происходит в следующей последовательности:

1) если в ячейке возврата текущей строки нет знака %, то err:=0;
2) если параметр единственный (нет ИЛИ элемента) или это последний элемент последовательности ИЛИ, и в ячейке возврата текущей строки нет знака %, то генерировать ошибку «Должен быть элемент % в позиции %%», где % либо терминальный символ (или его код), либо “идентификатор”, либо “литера”; %% – позиция в таблице выходных символов;
3) при наличии нескольких параметров (разделенных элементом ИЛИ) и если текущий не последний из них, то перейти на следующий параметр, более правый, значение переменной err присвоить значение 0;
4) если номер столбца текущей ячейки – 2, то если в ячейке возврата есть знак %, то перейти по адресу, указанному за знаком % и:

- в таблице переходов очистить ячейку возврата,
- в формируемой таблице переходов удалить последнюю строку.

2.4.3 Правила таблицы переходов для написания программы

Если ячейка пуста, то осуществляется переход на первую ячейку текущей строки, i:=1.

Если значение в ячейке типа x,y или x,y,z, то необходимо перейти на строку x, ячейку y, позицию z. При этом: после перехода в указанную ячейку на указанную позицию ячейка с адресом перехода, если она является ячейкой возврата, очищается (Table_Perehod[i,j]:=’’; i:=y; j:=x; pos:=z), в формируемой таблице переходов происходит переход в ячейку, указанную в первой ячейке строки без очищения ее значения (i1:=y; j1:=x).

Если значение в ячейке типа x,y%a,b,c, при этом err=1 и номер столбца равен 2 (i=2), то следует перейти по ссылке a,b,c, очистить ячейку возврата таблицы переходов (Table_Perehod[i,j]:=’’; i:=b; j:=a; pos:=c), в формируемой таблице переходов перейти по адресу возврата и удалить последнюю строку (i1:=y; j1:=x).

Если значение в ячейке типа x,y%a,b,c, и err=0, то перейти по ссылке x,y, в формируемой таблице переходов перейти по адресу, указанному в текущей ячейке.

Если номер столбца текущей ячейки = 3 и err<>0, то в ячейке возврата удалить при наличии знак % и значения за ним.

Если первый символ ^ – значение в ячейке является литерой (таблица литералов – №3). Осуществляемая при этом проверка: если в таблице выходных символов № текущей таблицы равен 3 (if Tabl_vs[count_vs,2]=’3’), то занести в текущую ячейку формируемой таблицы № таблицы (3) и № строки в ней (Table_Perehod1[i1,j1]:=$3,№строки), перейти на следующую ячейку (i:=i+1; i1:=i1+1; count_vs:=count_vs+1). В случае отрицательного результата сравнения переменной err присваивается значение 1.

Если первый символ $ – значение в ячейке является терминальным символом (таблица терминальных символов – №1). Осуществляемая проверка: если в таблице выходных символов № текущей таблицы равен 1 (if Tabl_vs[count_vs,2]=’1’), то занести в текущую ячейку формируемой таблицы № таблицы (1) и № строки в ней (Table_Perehod1[i1,j1]:=$1,код), перейти на следующую ячейку (i:=i+1; i1:=i1+1; count_vs:=count_vs+1). В случае отрицательного результата сравнения переменной err присваивается значение 1.

Если первый символ ~ – это переход на вторую ячейку строки с номером, указанным за символом ~, в формируемой таблице переходов добавляется новая строка и переход осуществляется на нее. При этом осуществляется следующее: в первую ячейку (ячейку возврата) указанной строки заносится адрес возврата: если переход осуществляется с одной из позиций с элементом ИЛИ и не является последним в списке, то в ячейке возврата формируется код возврата типа x,y,z, где x – номер строки, y – номер столбца, z – номер позиции откуда был произведен переход (x:=j; y:=i; z:=pos; j:=a; i:=2, где a номер строки в адресе перехода – ~a), тоже происходит и в формируемой таблицей переходов (x:=j1; y:=i1; j1:=№последней строки; i1:=2).

Коды терминальных символов показаны в таблице 11.

Таблица 11 – Таблица кодов терминальных символов

Код	Терминальный символ	Комментарий		
1	PROGRAM	объявление программу		
2	VAR	объявление переменных		
3	BEGIN	начало тела		
4	END	конец тела		
5	INTEGER	тип целое		
6	REAL	вещественный тип		
7	STRING	строковый тип		
8	FOR	цикл с параметром – ДЛЯ		
9	TO	цикл с параметром – ДО		
10	DO	ВЫПОЛНИТЬ		
11	REPEAT	цикл с постусловием – ПОВТОРЯТЬ		
12	UNTIL	цикл с постусловием – ПОКА НЕ		
13	WHILE	цикл с предусловием – ПОКА		
14	IF	условный оператор – ЕСЛИ		
15	THEN	условный оператор – ТО		
16	ELSE	условный оператор – ИНАЧЕ		
17	DIV	делить на цело		
18	WRITE	вывести на консоль		
19	READ	считать с консоли		
20	DOWNTO	цикл с параметром – ДО		
21	FUNCTION	функция		
22	PROCEDURE	процедура		
23	{	начало комментария		
24	}	конец комментария		
25	[	открытие квадратных скобок		
26	]	закрытие квадратных скобок		
27	;	конец операции		
28	:=	присвоить значение		
29	,	разделитель		
30	.	конец программы/отделение дробной части		
31	:	разделение идентификатора от его типа		
32	+	оператор сложения		
33	-	оператор вычитания		
34	*	оператор умножения		
35	(	открывающаяся скобка		
36	)	закрывающаяся скобка		
37	/	оператор деления		
38	′	кавычка		
39	<	меньше		
40	>	больше		
41	=	равно		
42	>=	больше или равно		
43	<=	меньше или равно		
44	<>	не равно		

2.4.4 Формируемая таблица переходов. Правила заполнения

Таблица представляет собой набор ячеек. Столбцы и строки нумеруются. Столбцы определяют номер распознанной лексемы (элемента конструкции), строки определяют номер полученной конструкции.

В таблице могут существовать только два вида данных: указатели на таблицы и переходы.

Указатели на таблицы состоят из символа признака ссылки на таблицу «$», номера таблицы и, через запятую, кода (для терминального символа) или спецификатора. Номера таблиц: 1– таблица терминальных символов, 2 – таблица символических имен, 3 – таблица литералов.

Указатели на ячейки состоят из символа «@» и следующих за ним через запятую адресов столбца и строки, куда следует перейти, оказавшись в данной ячейке.

Все ячейки первого столбца зарезервированы для значений ссылок обратного перехода на ячейку, следующую за вызвавшей переход в данную строку и называются ячейками возврата.

Чтобы заполнить формируемую таблицу переходов необходимо знать следующие правила.

Таблица заполняется в точности по БНФ, показанной на рисунке 4, при этом строки таблицы являются элементами конструкций данной грамматики. Разбор делается по данным, полученным из таблиц терминальных символов, символических имен, литералов, выходных кодов лексем.

1) Заполнение таблицы начинается с левой верхней ячейки, каждый раз смещаясь на следующую, более правую.

2) При наличии вложенности конструкции (найден нетерминальный символ) создается (заполняется) новая строка, где в первой ячейке указывается адрес возврата: указатель перехода «@», номер строки, номер столбца, откуда осуществлен переход плюс один (т.е. переход на следующую ячейку за той, из которой был произведен вызов).
3) При наличии нескольких элементов, разделенных знаком «|», осуществляется перебор вариантов, которые подходят значениям, указанным в таблице выходных кодов.
4) При соответствии значения ячейки или одного из элементов ИЛИ терминальному символу указывается признак ссылки на таблицу «$», затем номер таблицы – 1 и код элемента в таблице терминальных символов (номер таблицы и код берется из таблицы кодов лексем).
5) При соответствии значения ячейки или одного из элементов ИЛИ идентификатору указывается признак ссылки на таблицу «$», затем номер таблицы – 2 и спецификатор (номер таблицы и спецификатор берется из таблицы кодов лексем).
6) При соответствии значения ячейки или одного из элементов ИЛИ литералу указывается признак ссылки на таблицу «$», затем номер таблицы – 3 и спецификатор (номер таблицы и спецификатор также берется из таблицы кодов лексем).
7) Подчеркнутые элементы в БНФ грамматике не обязательны (они могут отсутствовать).

2.4.5 Правила заполнения формируемой таблицы переходов

Разбор делается по данным, полученным от БНФ грамматик, из таблиц терминальных символов, выходных кодов лексем.

Заполнение формируемой таблицы переходов начинается со второй ячейки первой строки.

Для терминальных символов алгоритм может выглядеть таким образом.

Допустим, производится анализ первого элемента таблицы кодов лексем.

1) Номер позиции в таблице кодов лексем равен 1. Производится чтение данных из первого столбца таблицы кодов лексем. Полученное значение номера таблицы – 1 (таблица терминальных символов), код равен 1 (по таблице кодов – «объявление программы»).
2) Рассматривается первый элемент БНФ грамматики, им является ключевое слово PROGRAM, оно принадлежит таблице терминальных символов – 1, по таблице кодов определяется код, он равен 1.
3) Производится сравнение номеров таблиц, полученных на этапе 1 и этапе 2. Значения совпали.
4) Производится сравнение кодов таблиц. Значения совпали.
5) Во вторую ячейку первой строки формируемой таблицы переходов заносится указатель на таблицу 1, код 1 – «$1,1».
6) Номер позиции в таблице кодов лексем увеличивается на 1 (стает равным 2).
7) В грамматике БНФ начинает рассматриваться следующий элемент конструкции – нетерминальный символ <prog-name>.
8) Новое значение в формируемую таблицу переходов будет заноситься в правую ячейку от текущей (номер столбца увеличился на 1).

При несовпадении значений на этапах 3 и 4 происходит прекращение разбора, если только терминальный символ не является элементов массива ИЛИ – «|» (например <type> ::= INTEGER | REAL | STRING) или не является необязательным элементом конструкции (например, PROGRAM <prog-name>), тогда осуществляется переход на следующий элемент массива ИЛИ или переход к

следующему элементу конструкции не входящему в данную группу необязательных элементов (подчеркиваются одной общей линией).

Для нетерминальных символов алгоритм примерно такой.

Допустим, производится анализ второго элемента таблицы кодов лексем. Текущая разбираемая лексема в грамматике БНФ является нетерминальным символом. В формируемой таблице переходов данные заносятся в третью ячейку первой строки.

1) Номер позиции в таблице кодов лексем равен 2. Производится чтение данных из второго столбца таблицы кодов лексем. Полученное значение номера таблицы – 2 (таблица символических имен), спецификатор равен 1.
2) Конструкция, определяющая структуру нетерминального символа <prog-name>, в грамматике БНФ имеет номер 2.
3) В формируемой таблице переходов добавляется новая строка, где в первую ячейку заносится адрес возврата на ячейку, вызвавшую переход, смещенную на 1 вправо (в данном случае указывается переход на четвертую ячейку первой строки – «@1,4»).
4) Номер позиции в таблице кодов лексем не изменяется (остается равным 2).
5) В грамматике БНФ начинает рассматриваться следующий элемент конструкции – **id** (идентификатор).
6) Новое значение в формируемую таблицу переходов будет заноситься в правую ячейку от текущей (в данном примере строка 2, столбец 2).

Для идентификаторов алгоритм примерно такой.

Допустим, производится анализ второго элемента таблицы кодов лексем. Текущая разбираемая лексема в грамматике БНФ находится в конструкции <prog-name> является первым ее элементом. В формируемой таблице переходов данные заносятся во вторую ячейку второй строки.

1) Номер позиции в таблице кодов лексем равен 2. Производится чтение данных из второго столбца таблицы кодов лексем. Полученное значение номера таблицы – 2 (таблица символических имен), спецификатор равен 1.
2) Рассматривается первый элемент БНФ грамматики конструкции <prog-name>, этот элемент является идентификатором, следовательно является элементом таблицы символических имен – таблицы 2.
3) Производится сравнение номеров таблиц. Значения совпали.
4) Во вторую ячейку второй строки заносится указатель на таблицу 2, спецификатор 1 (его значение берется из таблицы кодов лексем) – «$2,1»
5) Номер позиции в таблице кодов лексем увеличивается на 1 (стает равным 3).
6) В грамматике БНФ начинает рассматриваться следующий элемент конструкции <prog-name> – терминальный символ «;».
7) Новое значение в формируемую таблицу переходов будет заноситься в правую ячейку от текущей (номер столбца увеличился на 1) – вторая строка, третья ячейка.

При несовпадении значений на этапе 3, происходит прекращение разбора, если только текущий элемент не является элементов массива ИЛИ – «|» (например <factor> ::= **id** | **int** | **real** | <text-val> | (<exp>)) или не является необязательным элементом конструкции (например, PROGRAM <prog-name>), тогда осуществляется переход на следующий элемент массива ИЛИ или переход к следующему элементу конструкции не входящему в данную группу необязательных элементов (подчеркиваются одной общей линией).

Если конструкция закончилась, то осуществляется переход на первую ячейку (ячейку возврата) текущей строки в формируемой таблице перехода. По значению адреса, содержащегося в ячейке возврата активной (готовой для внесения данных) стает ячейка с указанным номером строки и номером столбца. В грамматике БНФ осуществляется переход к элементу, следующему за элемен-

том конструкции, который был определен в текущей конструкции. Например, после определения последнего элемента конструкции <prog-name> «;» осуществляется возврат к элементу, следующему за элементом, вызвавшим эту конструкцию. Возврат осуществлен на строку 1 грамматики БНФ, к элементу VAR, следующему за нетерминальным символом <prog-name>.

Для литералов алгоритм примерно такой.

Допустим, производится анализ шестнадцатого элемента таблицы кодов лексем. Текущая разбираемая лексема в грамматике БНФ находится в конструкции <factor> является вторым элементом конструкции ИЛИ. В формируемой таблице переходов данные заносятся во вторую ячейку десятой строки.

1) Номер позиции в таблице кодов лексем равен 16. Производится чтение данных из шестнадцатого столбца таблицы кодов лексем. Полученное значение номера таблицы – 3 (таблица литералов), спецификатор равен 1.
2) Рассматривается второй элемент множества ИЛИ БНФ грамматики конструкции <factor>, этот элемент является литералом целого типа, следовательно является элементом таблицы литералов – таблицы 3.
3) Производится сравнение номеров таблиц. Значения совпали.
4) Во вторую ячейку десятой строки заносится указатель на таблицу 3, спецификатор 1 (его значение берется из таблицы кодов лексем) – «$3,1»
5) Номер позиции в таблице кодов лексем увеличивается на 1 (стает равным 17).
6) В грамматике БНФ начинает рассматриваться следующий элемент конструкции <factor>, т.к. он отсутствует это говорит о том, что конец конструкции.
7) Вызывается обработка конца конструкции.

При несовпадении значений на этапе 3, происходит прекращение разбора, если только текущий элемент не является элементов массива ИЛИ – «|» (например <factor> ::= **id** | **int** | **real** | <text-val> | (<exp>)) или не является необязательным элементом конструкции (например, PROGRAM <prog-name>), тогда осуществляется переход на следующий элемент массива ИЛИ или переход к следующему элементу конструкции не входящему в данную группу необязательных элементов (подчеркиваются одной общей линией).

Описание работы с элементами массива ИЛИ БНФ грамматики.

Если в БНФ грамматике встречается массив ИЛИ, перебор значений осуществляется с левого.

При возникновении ошибки при работе с одним из элементов массива ИЛИ осуществляется переход к следующему, находящемуся правее.

В случае, когда последний (крайний правый) элемент массива ИЛИ или значение в ячейке не удовлетворительно, происходит прекращение разбора программы.

В случае положительного результата (сравнение одного из элементов удачно или сравнение с конечным результатом переходов удачно) происходит переход на следующую лексему грамматики БНФ, на следующий элемент таблицы кодов лексем, на следующую ячейку (на 1 правее) формируемой таблицы переходов.

Ниже рассматривается пример программы.

```
Program prog1;
var a,b,c:integer;
begin
  a:=1+b*(a–c);
end.
```

Полученная последовательность символов от программы LEXAN представлена в таблице 12.

Таблица 12 – Таблица выходных символов

№ п.п.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Таблица	1	2	1	1	2	1	2	1	2	1	1	1	1	2	1
Строка	1	1	27	2	2	29	3	29	4	31	5	27	3	2	28

№ п.п.	16	17	18	19	20	21	22	23	24	25	26	27
Таблица	3	1	2	1	1	2	1	2	1	1	1	1
Строка	1	32	3	34	35	2	33	4	36	27	4	30

По исходным данным, также используя таблицу кодов терминальных символов, заполняется таблица построений.

Для лучшего понимания заполнения формируемой таблицы переходов заполняется таблица построений. Обработка данных и заполнение таблицы процесс довольно трудоемкий, однако, надо понимать, что и для машины данный разбор является самой трудоемкой задачей. Здесь же процесс разбора программы максимально унифицирован. Дается возможность отследить все возможные (исходные) параметры (значения). Прервать заполнение таблицы построений можно в любой момент времени, а потом продолжить, не теряя логику переходов (построения).

Таблица построений состоит из нескольких разделов, они называются: шаги, таблица кодов лексем, имя в программе, элемент грамматики БНФ, результат сравнения, формируемая таблица переходов, выполненное действие.

В разделе «Шаги» перечисляются номера произведенных шагов. Раздел «Таблица кодов лексем» служит для отслеживания позиции в таблице кодов лексем и хранит значения таблицы и кода (или спецификатора) с которыми в дальнейшем происходит сравнение текущего элемента грамматики БНФ. Раздел «Элемент грамматики БНФ» содержит имя элемента, имя конструкции, в которую он входит, тип текущего элемента грамматики, номер таблицы, соответствующей ему, а также код (для терминальных символов), определенный по таблице кодов терминальных символов. В разделе «Формируемая таблица переходов» отслеживается текущая ячейка формируемой таблицы переходов, куда заносится формируемое значение, а также позиция следующей ячейки, с которой будет производиться работа на следующем шаге. В разделе «Выполненное действие» указывается выполненное действие.

Таблица построения предлагается как вспомогательное средство для заполнения формируемой таблицы переходов. Заполнять все шаги и ячейки в ней не обязательно, главное понять логику заполнения формируемой таблицы переходов. Приобретя опыт, в дальнейшем, можно обходиться без таблицы построений, заполняя формируемую таблицу переходов по БНФ грамматике, тексту программы, таблице кодов терминальных символов, таблице кодов лексем.

Из данных, полученных в разделе «Формируемая таблица переходов, текущая позиция» таблицы построения, заполняется формируемая таблица переходов. В результате должна получиться таблица 14.

Принятые сокращения в таблице построений.

НС – нетерминальный символ

ТС – терминальный символ

ИД – идентификатор

Лх – литерал, где х: Ц – целый тип, В – вещественный тип, С – строковый тип.

Л – литерал любого типа.

Наличие знака «–» впереди типа у элемента грамматики БНФ показывает, что данный элемент в разборе может не участвовать.

При заполнении таблицы построений особую сложность представляет работа с переходами. Ниже описывается работа с ними.

При обнаружении терминального символа в грамматике БНФ, необходимо осуществить переход на первый элемент конструкции с тем же именем. В таблице построений определяется номер последней не занятой строки в формируемой таблице переходов. Номером следующей позиции указывается номер этой строки, номер столбца – 1. Значение вносимого значения должно указывать на вторую ячейку последней пустой строки. Следующим шагом заполняется значение текущий позиции, адрес возврата и адрес следующей позиции. Например. После нахождения терминального символа <prog-name> начинает рассматриваться первый элемент конструкции <prog-name> – **id**. В формируемой таблице переходов последней свободной строкой является строка 2, на нее и осуществляется переход, следующая позиция указывает на строку 2, столбец 1 (шаг 2). На третьем шаге в ячейку возврата 2,1 (где 2 – номер строки, 1 – номер столбца) будет внесено значение, указывающее на ячейку 1,4, т.к. переход осуществлен из ячейки 1,3. Текущая позиция – 2,1, следующая позиция – 2,2.

При возврате возникают другие трудности. К примеру, при окончании конструкции происходит переход на пустую ячейку, затем осуществляется переход на ячейку возврата. Значение, заполненное в ячейке возврата ищется по таблице. По полученному значению осу-

ществляется переход. Например, при окончании конструкции <id-list> (шаг 20), текущей ячейкой оказывается ячейка 5,7. Затем производится переход в ячейку 5,1 (шаг 21). По таблице определяем, что адрес возврата @4,3 (значение из шага 14), т.е. перейти на четвертую строку, третий столбец.

Далее отыскивается положение в грамматике БНФ по имени предыдущей позиции. Например, после перехода в ячейку 4,3 (шаг 22) отыскиваем в таблице имя элемента грамматики ячейки 4,2 (значение из шага 13), им оказывается нетерминальный символ <id-list> конструкции <dec>. По грамматике БНФ определяется, что следующий элемент конструкции <dec> является «:».

Таблица 13 – Таблица построений

Шаги	Таблица кодов лексем				Имя в программе	Элемент грамматики БНФ					Результат сравнения	Формируемая таблица переходов					Выполненное действие
												текущая позиция			следующая позиция		
	позиция	табл	код, специф	тип		имя	текущая конструкция	тип	табл	код (для ТС)		строка	столбец	вносимое значение	строка	столбец	
1	1	1	1	ТС	PROGRAM	PROGRAM	<prog>	–ТС	1	1	+	1	2	$1,1	1	3	
2	2	2	1	ИД	Prog1	<prog-name>	<prog>	НС				1	3	@2,2	2	1	
3	2	2	1	ИД	Prog1							2	1	@1,4	2	2	
4	2	2	1	ИД	Prog1	id	<prog-name>	ИД	2		+	2	2	$2,1	2	3	
5	3	1	27	ТС	;	;	<prog-name>	–ТС	1	27	+	2	3	$1,27	2	4	
6	4	1	2	ТС	VAR		<prog-name>					2	4		2	1	конец конструкции
7	4	1	2	ТС	VAR							2	1		1	4	переход
8	4	1	2	ТС	VAR	VAR	<prog>	–ТС	1	2	+	1	4	$1,2	1	5	
9	5	2	2	ИД	a	<dec-list>	<prog>	НС				1	5	@3,2	3	1	
10	5	2	2	ИД	a							3	1	@1,6	3	2	
11	5	2	2	ИД	a	<dec>	<dec-list>	НС				3	2	@4,2	4	1	
12	5	2	2	ИД	a							4	1	@3,3	4	2	
13	5	2	2	ИД	a	<id-list>	<dec>	НС				4	2	@5,2	5	1	
14	5	2	2	ИД	a							5	1	@4,3	5	2	
15	5	2	2	ИД	a	id	<id-list>	ИД	2		+	5	2	$2,2	5	3	
16	6	1	29	ТС	,	,	<id-list>	ТС	1	29	+	5	3	$1,29	5	4	
17	7	2	3	ИД	b	id	<id-list>	ИД	2		+	5	4	$2,3	5	5	
18	8	1	29	ТС	,	,	<id-list>	ТС	1	29	+	5	5	$1,29	5	6	
19	9	2	4	ИД	c	id	<id-list>	ИД	2		+	5	6	$2,4	5	7	
20	10	1	31	ТС	:		<id-list>					5	7		5	1	конец конструкции
21	10	1	31	ТС	:							5	1		4	3	переход
22	10	1	31	ТС	:	:	<dec>	ТС	1	31	+	4	3	:	4	4	
23	11	1	5	ТС	INTEGER	<type>	<dec>	НС				4	4	@6,1	6	1	
24	11	1	5	ТС	INTEGER							6	1	@4,5	6	2	
25	11	1	5	ТС	INTEGER	INTEGER	<type>	ТС	1	5	+	6	2	$1,5	6	3	
26	12	1	27	ТС	;		<type>					6	3		6	1	конец конструкции
27	12	1	27	ТС	;							6	1		4	5	переход
28	12	1	27	ТС	;		<dec>					4	5		4	1	конец конструкции
29	12	1	27	ТС	;							4	1		3	3	переход
30	12	1	27	ТС	;	;	<dec-list>	–ТС	1	27	+	3	3	$1,27	3	4	
31	13	1	3	ТС	BEGIN		<dec-list>					3	4		3	1	конец конструкции
32	13	1	3	ТС	BEGIN							3	1		1	6	переход
33	13	1	3	ТС	BEGIN	BEGIN	<prog>	ТС	1	3	+	1	6	$1,3	1	7	
34	14	2	2	ИД	a	<stmt-list>	<prog>	–НС				1	7	@7,2	7	1	
35	14	2	2	ИД	a							7	1	@1,8	7	2	
36	14	2	2	ИД	a	<stmt>	<stmt-list>	НС				7	2	@8,2	8	1	
37	14	2	2	ИД	a							8	1	@7,3	8	2	
38	14	2	2	ИД	a	<assign>	<stmt>	НС				8	2	@9,2	9	1	
39	14	2	2	ИД	a							9	1	@8,3	9	2	

Продолжение таблицы 13

Шаги	Таблица кодов лексем				Имя в программе	Элемент грамматики БНФ					Результат сравнения	Формируемая таблица переходов					Выполненное действие
												текущая позиция			следующая позиция		
	позиция	табл	код, специф	тип		имя	текущая конструкция	тип	табл	код (для ТС)		строка	столбец	вносимое значение	строка	столбец	
40	14	2	2	ИД	a	id	<assign>	ИД	2		+	9	2	$2,2	9	3	
41	15	1	28	ТС	:=	:=	<assign>	ТС	1	28	+	9	3	$1,28	9	4	
42	16	3	1	ЛЦ	1	<exp>	<assign>	НС				9	4	@10,2	10	1	
43	16	3	1	ЛЦ	1							10	1	@9,5	10	2	
44	16	3	1	ЛЦ	1	–	<exp>	–ТС	1	33	–	10	2				
45	16	3	1	ЛЦ	1	<term>	<exp>	НС				10	2	@11,2	11	1	
46	16	3	1	ЛЦ	1							11	1	@10,3	11	2	
47	16	3	1	ЛЦ	1	<factor>	<term>	НС				11	2	@12,2	12	1	
48	16	3	1	ЛЦ	1							12	1	@11,3	12	2	
49	16	3	1	ЛЦ	1	id	<factor>	ИД	2		–	12	2				
50	16	3	1	ЛЦ	1	int	<factor>	ЛЦ	3		+	12	2	$3,1	12	3	
51	17	1	32	ТС	+		<factor>					12	3		12	1	конец конструкции
52	17	1	32	ТС	+							12	1		11	3	переход
53	17	1	32	ТС	+	*	<term>	ТС	1	34	–	11	3				
54	17	1	32	ТС	+	DIV	<term>	ТС	1	17	–	11	3				
55	17	1	32	ТС	+	/	<term>	ТС	1	37	–	11	3				
56	17	1	32	ТС	+		<term>					11	3		11	1	конец конструкции
57	17	1	32	ТС	+							11	1		10	3	переход
58	17	1	32	ТС	+	+	<exp>	ТС	1	32	+	10	3	$1,32	10	4	
59	18	2	3	ИД	b	<term>	<exp>	НС				10	4	@13,2	13	1	
60	18	2	3	ИД	b							13	1	@10,5	13	2	
61	18	2	3	ИД	b	<factor>	<term>	НС				13	2	@14,2	14	1	
62	18	2	3	ИД	b							14	1	@13,3	14	2	
63	18	2	3	ИД	b	id	<factor>	ИД	2		+	14	2	$2,3	14	3	
64	19	1	34	ТС	*		<factor>					14	3		14	1	конец конструкции
65	19	1	34	ТС	*							14	1		13	3	переход
66	19	1	34	ТС	*	*	<term>	ТС	1	34	+	13	3	$1,34	13	4	
67	20	1	35	ТС	(	<factor>	<term>					13	4	@15,2	15	1	
68	20	1	35	ТС	(		<term>					15	1	@13,5	15	2	
69	20	1	35	ТС	(	id	<factor>	ИД	2		–	15	2				
70	20	1	35	ТС	(	int	<factor>	ЛЦ	3		–	15	2				
71	20	1	35	ТС	(	real	<factor>	ЛВ	3		–	15	2				
72	20	1	35	ТС	(	(	<factor>	ТС	1	35	+	15	2	$1,35	15	3	
73	21	2	2	ИД	a	<exp>	<factor>	НС				15	3	@16,2	16	1	
74	21	2	2	ИД	a							16	1	@15,4	16	2	
75	21	2	2	ИД	a	–	<exp>	–ТС	1	33	–	16	2				
76	21	2	2	ИД	a	<term>	<exp>	НС				16	2	@17,2	17	1	
77	21	2	2	ИД	a							17	1	@16,3	17	2	
78	21	2	2	ИД	a	<factor>	<term>	НС				17	2	@18,2	18	1	

Продолжение таблицы 13

Шаги	Таблица кодов лексем				Имя в программе	Элемент грамматики БНФ					Результат сравнения	Формируемая таблица переходов					Выполненное действие
												текущая позиция			следующая позиция		
	позиция	табл	код, специф	тип		имя	текущая конструкция	тип	табл	код (для ТС)		строка	столбец	вносимое значение	строка	столбец	
79												18	1	@17,3	18	2	
80	21	2	2	ИД	a	id	<factor>	ИД	2		+	18	2	$2,2	18	3	
81	22	1	33	ТС	–							18	3		18	1	конец конструкции
82	22	1	33	ТС	–							18	1		17	3	переход
83	22	1	33	ТС	–	*	<term>	ТС	1	34	–	17	3				
84	22	1	33	ТС	–	DIV	<term>	ТС	1	17	–	17	3				
85	22	1	33	ТС	–	/	<term>	ТС	1	37	–	17	3				
86	22	1	33	ТС	–							17	3		17	1	конец конструкции
87	22	1	33	ТС	–							17	1		16	3	переход
88	22	1	33	ТС	–	+	<exp>	ТС	1	32	–	16	3				
89	22	1	33	ТС	–	–	<exp>	ТС	1	33	+	16	3	$1,33	16	4	
90	23	2	4	ИД	c	<term>	<exp>	НС				16	4	@19,2	19	1	
91	23	2	4	ИД	c							19	1	@16,5	19	2	
92	23	2	4	ИД	c	<factor>	<term>	НС				19	2	@20,2	20	1	
93	23	2	4	ИД	c							20	1	@19,3	20	2	
94	23	2	4	ИД	c	id	<factor>	ИД	2		+	20	2	$2,4	20	3	
95	24	1	36	ТС	)		<factor>					20	3		20	1	конец конструкции
96	24	1	36	ТС	)							20	1		19	3	переход
97	24	1	36	ТС	)	*	<term>				–	19	3				
98	24	1	36	ТС	)	DIV	<term>				–	19	3				
99	24	1	36	ТС	)	/	<term>				–	19	3				
100	24	1	36	ТС	)		<term>					19	3		19	1	конец конструкции
101	24	1	36	ТС	)							19	1		16	5	переход
102	24	1	36	ТС	)	+	<exp>	ТС	1	32	–	16	5				
103	24	1	36	ТС	)	–	<exp>	ТС	1	33	–	16	5				
104	24	1	36	ТС	)		<exp>					16	5		16	1	конец конструкции
105	24	1	36	ТС	)							16	1		15	4	переход
106	24	1	36	ТС	)	)	<factor>	ТС	1	36	+	15	4	$1,36	15	5	
107	25	1	27	ТС	;		<factor>					15	5		15	1	конец конструкции
108	25	1	27	ТС	;							15	1		13	5	переход
109	25	1	27	ТС	;	*	<term>	ТС	1	34	–	13	5				
110	25	1	27	ТС	;	DIV	<term>	ТС	1	17	–	13	5				
111	25	1	27	ТС	;	/	<term>	ТС	1	37	–	13	5				
112	25	1	27	ТС	;							13	5		13	1	конец конструкции
113	25	1	27	ТС	;							13	1		10	5	переход
114	25	1	27	ТС	;	+	<exp>	ТС	1	32		10	5				
115	25	1	27	ТС	;	–	<exp>	ТС	1	33		10	5				
116	25	1	27	ТС	;							10	5		10	1	конец конструкции
117	25	1	27	ТС	;							10	1		9	5	переход

Продолжение таблицы 13

Шаги	Таблица кодов лексем				Имя в программе	Элемент грамматики БНФ					Результат сравнения	Формируемая таблица переходов					Выполненное действие
												текущая позиция			следующая позиция		
	позиция	табл	код, специф	тип		имя	текущая конструкция	тип	табл	код (для ТС)		строка	столбец	вносимое значение	строка	столбец	
118	25	1	27	ТС	;		<assign>					9	5		9	1	конец конструкции
119	25	1	27	ТС	;							9	1		8	3	переход
120	25	1	27	ТС	;		<stmt>					8	3		8	1	конец конструкции
121	25	1	27	ТС	;							8	1		7	3	переход
122	25	1	27	ТС	;	;	<stmt-list>	–ТС	1	27	+	7	3	$1,27	7	4	
123	26	1	4	ТС	END		<stmt-list>					7	4		7	1	конец конструкции
124	26	1	4	ТС	END							7	1		1	8	переход
125	26	1	4	ТС	END	END	<prog>	ТС	1	4	+	1	8	$1,4	1	9	
126	27	1	30	ТС	.	.	<prog>	ТС	1	30	+	1	9	$1,30	1	10	
127												1	10		1	1	конец конструкции
128												1	1				

Таблица 14 – Формируемая таблица переходов

		1	2	3	4	5	6	7	8	9	10
1			PROGRAM $1,1	<prog-name> @2,2	VAR $1,2	<dec-list> @3,2	BEGIN $1,3	<stmt-list> @7,2	END $1,4	. $1,30	
2	<prog-name>	@1,4	prog1 $2,1	; $1,27							
3	<dec-list>	@1,6	<dec> @4,2	; $1,27							
4	<dec>	@3,3	<id-list> @5,2	: $1,31	<type> @6,2						
5	<id-list>	@4,3	a $2,2	, $1,29	b $2,3	, $1,29	c $2,4				
6	<type>	@4,5	INTEGER $1,5								
7	<stmt-list>	@1,8	<stmt> @8,2	; $1,27							
8	<stmt>	@7,3	<assign> @9,2								
9	<assign>	@8,3	a $2,2	:= $1,28	<exp> @10,2						
10	<exp>	@9,5	<term> @11,2	+ $1,32	<term> @13,2						
11	<term>	@10,3	<factor> @12,2								
12	<factor>	@11,3	1 $3,1								
13	<term>	@10,5	<factor> @14,2	* $1,34	<factor> @15,2						
14	<factor>	@13,3	b $2,3								
15	<factor>	@13,5	($1,35	<exp> @16,2	) $1,36						
16	<exp>	@15,4	<term> @17,2	– $1,33	<term> @19,2						
17	<term>	@16,3	<factor> @18,2								
18	<factor>	@17,3	a $2,2								
19	<term>	@16,5	<factor> @20,2								
20	<factor>	@19,3	c $2,4								

2.4.6 Построение деревьев

Для наглядного отображения полученных грамматик используют синтаксические деревья, пример показан на рисунке 3.

На основе введенных значений формируемой таблицы переходов в программу SINAN строится (формируется) синтаксическое дерево (дерево грамматического разбора).

Деревья могут быть представлены в вертикально как показано на рисунке 5, а и горизонтально – рисунок 5, б.

Рассмотрим выражение: a := c – (1 + b)

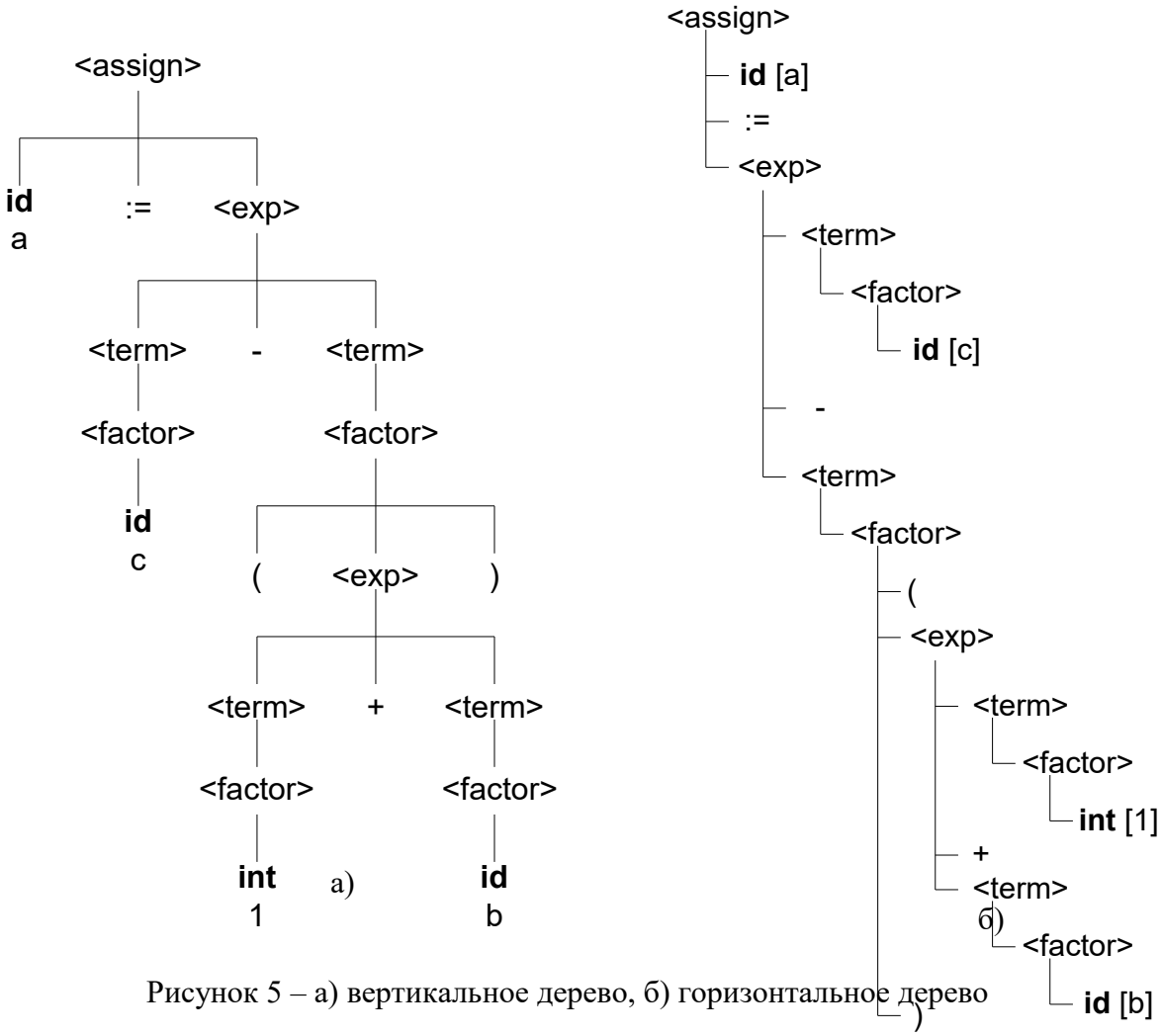

Рисунок 5 – а) вертикальное дерево, б) горизонтальное дерево

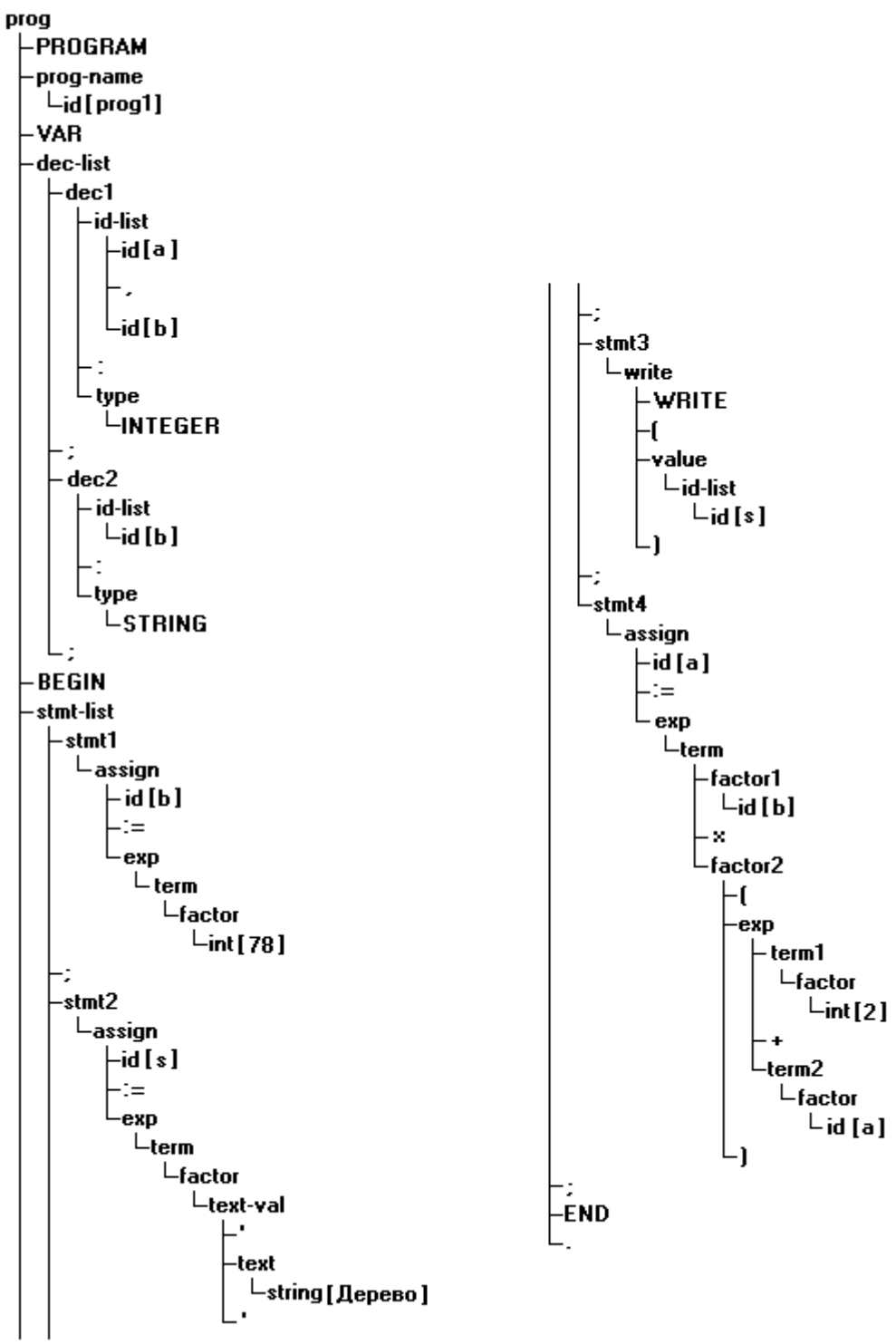

Рисунок 6 – Горизонтальное синтаксическое дерево

На рисунке 6 показано горизонтальное синтаксическое дерево, построенное по следующей программе:

```
PROGRAM prog1;
VAR a,b:INTEGER;
      s:STRING;
BEGIN
b:=78;
s:='Дерево';
WRITE(s);
a:=b*(2+a);
END.
```

2.4.7 Семантический анализ

Функции семантического анализатора:

1) ведение табличных символов;
2) включение неявной информации (по умолчанию);
3) обнаружение ошибок;
4) макрообработка и операции, выполняемые во время компиляции.

После проведения синтаксического анализа формируется дерево грамматического разбора, представленное в виде таблицы. По этому дереву на этапе семантического анализа производится новый смотр.

Одно из предназначений семантического анализатора – поиск ошибок. Существуют следующие критерии поиска ошибок:

1) не должно быть повторного описания идентификатора;
2) все идентификаторы, используемые в программе, должны быть описаны;
3) запрещается присвоение значению переменной одного типа значение другого типа (возможно только присвоение вещественному типу целого значения);
4) результат деления “ / “ всегда вещественное число;
5) перед использованием переменной (идентификатора) ей должно быть присвоено значение (данная ошибка не относится к критическим).

6) в цикле FOR, структуре <index-exp>, идентификатор должен быть целого типа, как и оба значения возвращаемые структурой <exp>.

Пример неправильно написанных элементов программы.

```
VAR
    A,B,C:INTEGER;
    C,D:REAL                    – повторное описание переменной C
BEGIN
    A:=3.5;                     – присвоение переменной целого типа вещественного значения
    B:=A/2;                     – присвоение переменной целого типа вещественного значения, образующегося при делении
    D:=F*5;                     – не описана переменная F
    FOR A:=1 TO D DO C:=C+A     – переменная D вещественного типа
END.
```

2.5 Формирование промежуточного кода

Промежуточный код может быть представлен в виде польского кода или тетрад. Так как в данной работе не используется стек, следовательно польская форма записи не подходит. Принимаем за основу метод четверок (тетрады).

Разница между этими методами в том, что результат каждого из этапов в тетрадах представляется отдельной промежуточной переменной, в связи с чем требуется большое количество памяти на хранение этих переменных. Представление промежуточных данных кроме всего имеет большую наглядность по этому за основу был принят метод четверок (тетрад).

Циклы

При адресации используются следующие команды.

C $BR – безусловный переход на позицию (индекс) в массиве, содержащем тетрады.

L $BRL – безусловный переход на метку

L - имя в таблице символов. Значение его - адрес перехода. Основная проблема при реализации этого оператора – определение адреса перехода.

<операнд1> <операнд2> BRZ|$BRM|$BRP|$BRMZ|$BRPZ

<операнд1> - значение арифметического выражения,

<операнд2> - номер или место позиции (адрес).

$BRZ - переход по значению 0,

$BRM - переход по значению <0,

$BRP - переход по значению >0,

$BRMZ - переход по значению <=0,

$BRPZ - переход по значению >=0.

Реализация циклов не вызывает сложностей. Имея оператор безусловного перехода и условный оператор, можно сконструировать цикл "вручную". Например, цикл вида

FOR I=N1 TO N2 DO operator

может быть сконструирован на исходном языке:

I := N1;

L1: IF I>N2 THEN GOTO L2;

operator;

I:=I+1;

GOTO L1;

L2:

Представление конструкции FOR I=N1 TO N2 DO operator в виде тетрад показано в таблице 15.

Таблица 15 – конструкция for в виде тетрад

Выражения	Тетрады (четверки)			
I:=N1	:=	N1		I
L1				
IF I>N2 THEN GOTO L2 (IF N2-I<0 THEN GOTO L2)	– $BRM	N2 L2	I T1	T1
operator	operator			
I:=I+1	+	I	1	I
GOTO L1	$BR	L1		
L2				

Далее рассмотрены циклы WHILE, REPEAT, а также конструкция IF…THEN…ELSE.

В таблице 16 разобрана конструкция WHILE a<b DO operator.

Таблица 16 – Конструкция while

Выражения	Тетрады (четверки)			
L1				
IF a-b>0 THEN GOTO L2	– $BRP	a L2	b T1	T1
operator	operator			
GOTO L1	$BR	L1		
L2				

В таблице 17 разобрана конструкция REPEAT operator UNTIL a<b

Таблица 17 – Конструкция repeat

Выражения	Тетрады (четверки)			
L1				
operator	operator			
IF a-b<0 THEN GOTO L1	– $BRM	a L1	b T1	T1

В таблице 18 разобрана конструкция IF a>b THEN operator1 ELSE operator2.

Таблица 18 – Конструкция if

Выражения	Тетрады (четверки)			
L1				
IF a-b<0 THEN GOTO L2	– $BRM	a L2	b T1	T1
operator1	operator1			
GOTO L3	$BR	L1		
L2				
operator2	operator2			
L3				

Контрольные вопросы (ПК-11):

1. Какова цель семантического анализатора?
2. Почему распределение памяти не может быть выполнено до выполнения семантического анализа?
3. Можно ли построить компилятор без оптимизации кода?
4. В чем заключается процедура генерации кода?
5. Какие способы внутреннего представления программ существуют?
6. Как образуется обратная польская запись операций?
7. Как вычисляются выражения с помощью обратно польской записи?
8. Как представляются в алгоритме операторы и операнды?
9. Какие принципы лежат в основе распределения памяти?
10. Почему в компиляторах используют относительные адреса памяти?