Лексический анализатор

Глава 5

Глава 5
Лексический анализатор

5.1. Назначение лексического анализатора

5.2. Принцип работы лексического анализатора

5.3. Организация взаимосвязи лексического и синтаксического анализатора

5.4. Таблица лексем

5.5. Принципы построения лексических анализаторов

5.6. Программная реализация лексического анализатра

5.1. Назначение лексического анализатора

Первая фаза компиляции называется лексическим анализом или сканированием. Лексический анализатор (сканер) читает поток символов, составляющих исходную программу, и группирует эти символы в значащие последовательности, называющиеся **лексемами**.

Важно!!! Какие объекты считать лексемами, зависит от входного языка программирования.

Определение. **Лексема** – это структурная единица языка, которая состоит из элементарных символов языка и не содержит в своём составе других структурных единиц языка.

Лексемами языков программирования являются идентификаторы, константы, ключевые слова языка, знаки операций и т.п.

5.1. Назначение лексического анализатора

На вход лексического анализатора поступает текст исходной программы, а выходная информация передаётся для дальнейшей обработки синтаксическому анализатору.

Для каждой лексемы сканер строит выходной токен (англ. token) вида

‹имя_токена, значение_атрибута›

Первый компонент токена, имя_токена, представляет собой абстрактный символ, использующийся во время синтаксического анализа, а второй компонент, значение атрибута, указывает на запись в таблице идентификаторов, соответствующую данному токену.

Лексический анализатор – это транслятор, входом которого служит цепочка символов, представляющая программу, а выходом – последовательность лексем. Этот выход образует вход синтаксического анализатора.

5.1. Назначение лексического анализатора

На вход лексического анализатора поступает текст исходной программы, а выходная информация передаётся для дальнейшей обработки синтаксическому анализатору.

Для каждой лексемы сканер строит выходной токен (англ. token) вида

‹имя_токена, значение_атрибута›

Первый компонент токена, имя_токена, представляет собой абстрактный символ, использующийся во время синтаксического анализа, а второй компонент, значение атрибута, указывает на запись в таблице идентификаторов, соответствующую данному токену.

Лексический анализатор – это транслятор, входом которого служит цепочка символов, представляющая программу, а выходом – последовательность лексем. Этот выход образует вход синтаксического анализатора.

5.1. Назначение лексического анализатора

С теоретической точки зрения ЛА не является обязательной частью компилятора. Все его функции могут выполняться на этапе синтаксического анализа, поскольку полностью регламентированы синтаксисом входного языка.

Тем не менее, в состав практически всех компиляторов включают лексический анализ по следующим причинам:

- применение ЛА упрощает работу с текстом исходной программы на этапе синтаксического разбора и сокращает объем обрабатываемой информации;
- для выделения в тексте и разбора лексем возможно применять простую, эффективную и теоретически хорошо проработанную технику анализа, в то время как на этапе синтаксического анализа конструкций исходного языка используются достаточно сложные алгоритмы разбора;
- ЛА отделяет сложный по конструкции синтаксический анализатор от работы непосредственно с текстом исходной программы, структура которого может варьироваться в зависимости от версии входного языка.

5.1. Назначение лексического анализатора

Замечание 1. При такой конструкции компилятора для перехода от одной версии языка к другой достаточно только перестроить относительно простой лексический анализатор.

Замечание 2. Лексический анализатор структурирует поступающий на вход исходный текст программы и выкидывает всю незначащую информацию.

5.1. Назначение лексического анализатора

Пример. Исходная программа содержит инструкцию присваивания

$$a = b + c * d$$

Символы в этом присваивании могут быть сгруппированы в следующие лексемы и отображены в следующие токены:

1) *a* представляет собой лексему, которая может отображаться в токен ‹*id*, 1›, где id – абстрактный символ, обозначающий идентификатор, а 1 указывает запись в таблице идентификаторов для *a*, в которой хранится такая информация как имя и тип идентификатора.
2) Символ присваивания = представляет собой лексему, которая отображается в токен ‹=›. Поскольку этот токен не требует значения атрибута, второй компонент данного токена опущен. В качестве имени токена может быть использован любой абстрактный символ, например такой, как «assign», но для удобства записи в качестве имени абстрактного символа можно использовать саму лексему.

5.1. Назначение лексического анализатора

3) *b* представляет собой лексему, которая отображается в токен ‹*id*, 2›, где 2 указывает на запись в таблице идентификаторов для *b*.
4) + является лексемой, отображаемой в токен ‹+›.
5) *c* – лексема, отображаемая в токен ‹*id*, 3›, где 3 указывает на запись в таблице идентификаторов для *c*.
6) * – лексема, отображаемая в токен ‹*›.
7) *d* – лексема, отображаемая в токен ‹*id*, 4›, где 4 указывает на запись в таблице идентификаторов для *d*.

Пробелы, разделяющие лексемы, лексическим анализатором пропускаются. Представление инструкции присваивания после лексического анализа в виде последовательности токенов примет следующий вид:

‹id, 1›‹=›‹id, 2›‹+›‹›‹id, 3›‹id, 4›.*

5.1. Назначение лексического анализатора

В таблице 1 приведены некоторые типичные токены, неформальное описание их шаблонов и некоторые примеры лексем.

Определение. Шаблон (англ. pattern) – это описание вида, который может принимать лексема токена.

В случае ключевого слова шаблон представляет собой просто последовательность символов, образующих это ключевое слово. Чтобы увидеть использование этих концепций на практике, рассмотрим инструкцию на языке программирования Си

printf("Total = %d\n", score);

в которой *printf* и *score* представляют собой лексемы, соответствующие токену *id*,

а *"Total = %d\n"* является лексемой, соответствующей токену *literal*.

5.1. Назначение лексического анализатора

Таблица 1. Примеры токенов

Токен	Неформальное описание	Примеры лексем
if	Символы i, f	`if`
else	Символы e, l, s, e	`else`
comp	< или > или <= или >= или == или !=	`<=`
id	Буква, за которой следуют буквы и цифры	`score, D2`
number	Любая числовая константа	`3.14159`
literal	Последовательность любых символов, заключённая в кавычки (кроме самих кавычек)	`"Total = %d\n"`

5.2. Принцип работы лексического анализатора

Функции, выполняемые лексическим анализатором, и состав лексем, которые он выделяет в тексте исходной программы, могут меняться в зависимости от версии компилятора.

В основном лексические анализаторы выполняют исключение из текста исходной программы комментариев и незначащих пробелов, а также выделение лексем следующих типов:

идентификаторов,

строковых, символьных и числовых констант,

ключевых (служебных) слов входного языка

5.2. Принцип работы лексического анализатора

Основные функции работы лексического анализатора:

- исключить из текста исходной программы комментарии, незначащие пробелы, символы табуляции, перевода строки;
- выделить лексемы следующих типов: идентификаторы, строковые, символьные и числовые константы, ключевые (*служебные*) слова входного языка, знаки операций и разделителей.
- четко определить границы лексемы, которые в исходном тексте явно не заданы;
- выполнить действия для сохранения информации об обнаруженной лексеме *(или выдать сообщение об ошибке, если лексема неверна*).

5.3. Организация взаимосвязи лексического и синтаксического анализатора

В большинстве компиляторов лексический и синтаксический анализаторы – это взаимосвязанные части.

Лексический разбор исходного текста в таком варианте выполняется поэтапно, так что синтаксический анализатор, выполнив разбор очередной конструкции языка, обращается к сканеру за следующей лексемой.

При этом он может сообщить информацию о том, какую лексему следует ожидать. В процессе разбора может даже происходить «откат назад», чтобы выполнить анализ текста на другой основе.

В дальнейшем будем исходить из предположения, что все лексемы могут быть однозначно выделены сканером на этапе лексического разбора.

5.3. Организация взаимосвязи лексического и синтаксического анализатора

Работу синтаксического и лексического анализаторов можно изобразить в виде схемы, приведённой на рисунке 1.

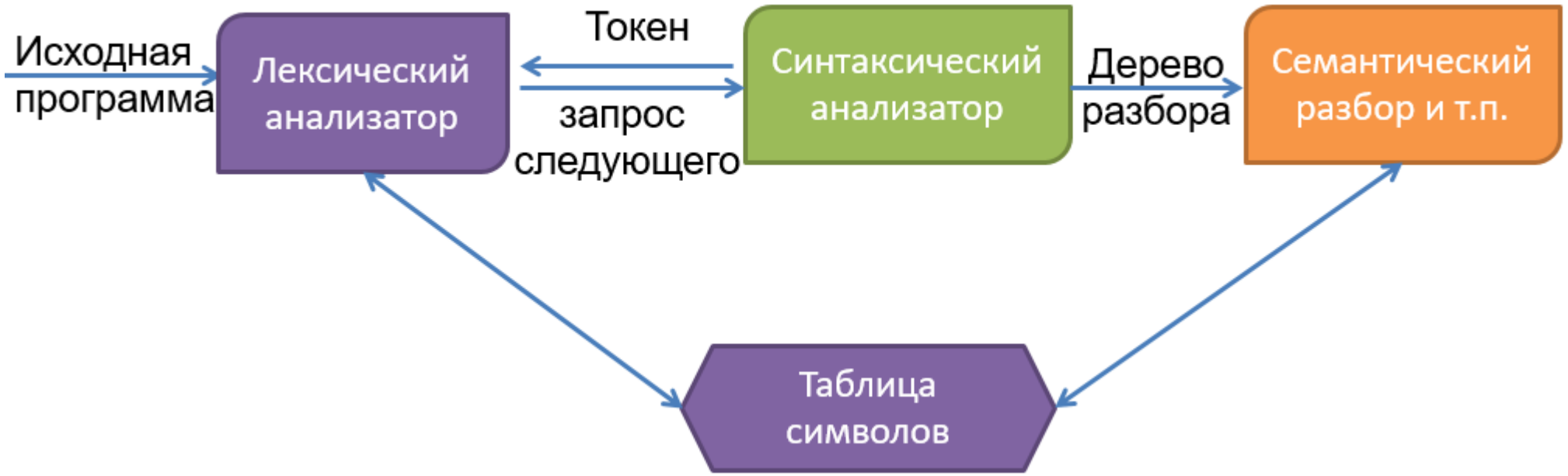

Рисунок 1. Взаимодействие лексического анализатора с синтаксическим

5.3. Организация взаимосвязи лексического и синтаксического анализатора

Последовательная – сканер просматривает весь текст исходной программы от начала до конца и преобразует его в структурированный набор данных (***таблицу лексем***), в которой ключевые слова языка, идентификаторы и константы, как правило, заменяются на специально оговоренные коды, им соответствующие.

Для идентификаторов и констант, кроме того, устанавливается связь между *таблицей лексем* и *таблицей идентификаторов*, которая заполняется параллельно.

Параллельная – лексический анализ исходного текста выполняется поэтапно так, что синтаксический анализатор, выполнив разбор очередной конструкции языка, обращается к ЛА за следующей лексемой.

При этом он может сообщить информацию о том, какую лексему следует ожидать.

5.3. Организация взаимосвязи лексического и синтаксического анализатора

Рисунок 1.1 Последовательное взаимодействие лексического и синтаксического анализаторов

5.3. Организация взаимосвязи лексического и синтаксического анализатора

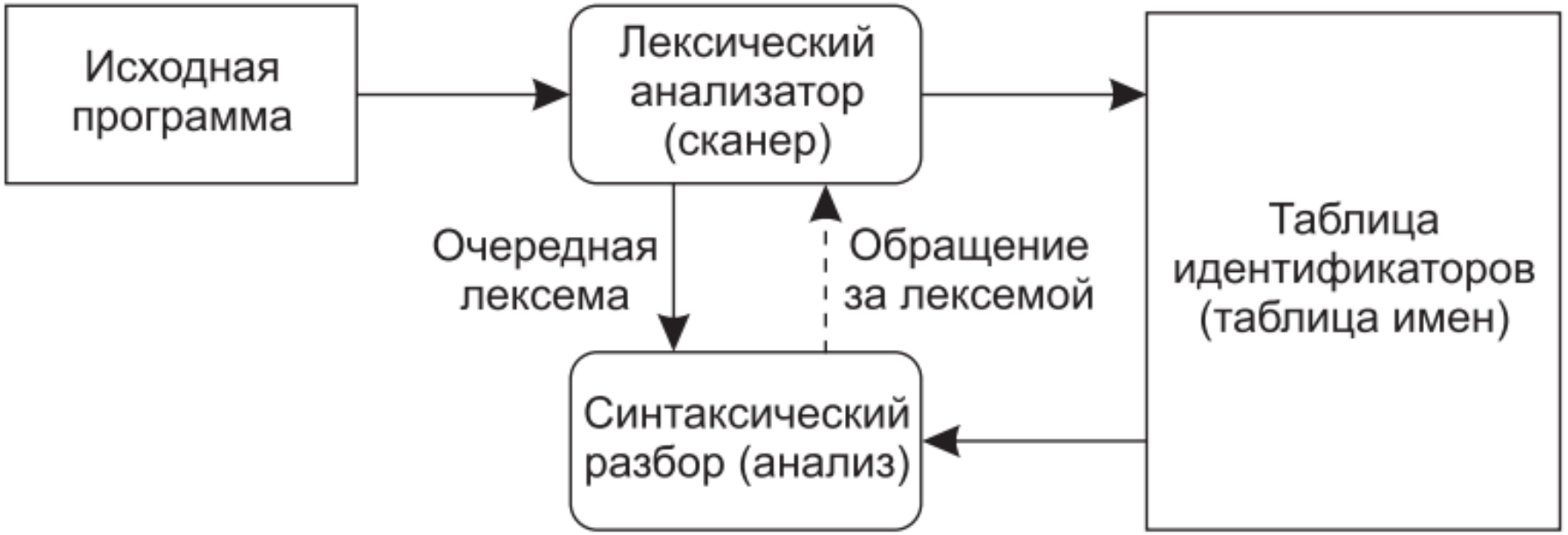

Рисунок 1.2 Параллельное взаимодействие лексического и синтаксического анализаторов

5.3. Организация взаимосвязи лексического и синтаксического анализатора

Пример. Иллюстрация последовательной реализации.

Оператор присваивания из языка программирования Си

k = i+++++j;

Данный оператор имеет только одну верную интерпретацию (если операции разделить пробелами):

k = i++ + ++j;

5.3. Организация взаимосвязи лексического и синтаксического анализатора

Замечание 3. При последовательной реализации ЛА просматривает весь текст исходной программы один раз от начала до конца. Таблица лексем строится полностью вся сразу, и больше к ней компилятор не возвращается. Всю дальнейшую обработку выполняют следующие фазы компиляции.

Замечание 4. В параллельном варианте реализации при возникновении ошибки может происходить «откат назад» (для попытки выполнения анализа текста на другой основе). В случае успешного выполнения разбора очередной конструкции языка синтаксическим анализатором, лексический анализатор помещает найденные лексемы в таблицу лексем и таблицу идентификаторов и анализ продолжается дальше в том же порядке.

5.3. Организация взаимосвязи лексического и синтаксического анализатора

Важно!!!. Очевидно, что параллельная работа лексического и синтаксического анализаторов более сложна в реализации, чем их последовательное выполнение. Кроме того, такая реализация требует больше вычислительных ресурсов и в общем случае большего времени на анализ исходной программы.

5.3. Организация взаимосвязи лексического и синтаксического анализатора

Чтобы избежать параллельной работы лексического и синтаксического анализаторов, разработчики компиляторов и языков программирования часто идут на разумные ограничения синтаксиса входного языка.

Например, для языка Си принято соглашение, что при возникновении проблем с определением границ лексемы всегда выбирается лексема максимально возможной длины.

5.3. Организация взаимосвязи лексического и синтаксического анализатора

Пример. Иллюстрация параллельной реализации.

Оператор присваивания из языка программирования Си

k = i+++++j;

Для данного оператора присваивания это приводит к тому, что при чтении четвёртого знака + из двух вариантов лексем (+ – знак сложения, ++ – оператор инкремента) лексический анализатор выбирает самую длинную, т.е. ++, и в целом весь оператор будет разобран как

k = i++ ++ +j;

что неверно.

5.3. Организация взаимосвязи лексического и синтаксического анализатора

Компилятор *gcc* в этом случае выдаст сообщение об ошибке:

lvalue required as increment operand – в качестве операнда оператора инкремента требуется *l*-значение.

Любые неоднозначности при анализе данного оператора присваивания могут быть исключены только в случае правильной расстановки пробелов в исходной программе.

5.4. Таблица лексем

Вид представления информации после выполнения лексического анализа целиком зависит от конструкции компилятора.

Но в общем виде её можно представить как **таблицу лексем**, которая в каждой строчке должна содержать информацию о виде лексемы, её типе и, возможно, значении.

Обычно такая таблица имеет следующие столбцы:

первый – строка лексемы,

второй – указатель на информацию о лексеме, может быть включён

третий – тип лексем.

Важно!!! Не следует путать таблицу лексем и таблицу идентификаторов – это две принципиально разные таблицы!

5.4. Таблица лексем

Таблица лексем содержит весь текст исходной программы, обработанный лексическим анализатором. В неё входят все возможные типы лексем, при этом, любая лексема может в ней встречаться любое число раз.

Таблица идентификаторов содержит только следующие типы лексем: идентификаторы и константы. В неё не попадают ключевые слова входного языка, знаки операций и разделители.

Важно!!! Каждая лексема в таблице идентификаторов может встречаться только один раз.

5.4. Таблица лексем

Пример. Фрагмента кода программы на языке Pascal и соответствующая ему таблица лексем

```
begin
    for i:=1 to N do
        fg := fg * 0.5
```

5.4. Таблица лексем

Таблица 2. Таблица лексем программы

Лексема	Тип лексемы	Значение
`begin`	Ключевое слово	X1
`for`	Ключевое слово	X2
`i`	Идентификатор	i : 1
`:=`	Знак присваивания	
`1`	Целочисленная константа	1
`to`	Ключевое слово	X3
`N`	Идентификатор	N : 2
`do`	Ключевое слово	X4
`fg`	Идентификатор	fg : 3
`:=`	Знак присваивания	
`fg`	Идентификатор	fg : 3
`*`	Знак арифметической операции	
`0.5`	Вещественная константа	0.5

5.4. Таблица лексем

В таблице 2 поле «*значение*» подразумевает некое кодовое значение, которое будет помещено в итоговую таблицу лексем.

Значения, приведённые в примере, являются условными. Конкретные коды выбираются разработчиками при реализации компилятора.

Связь между таблицей лексем и таблицей идентификаторов отражена в примере некоторым индексом, следующим после идентификатора за знаком «:».

В реальном компиляторе эта связь определяется его реализацией.

Пример:

```
begin
        if a>b then a:=a-b
                else a:=b*0.3;
end;
```

Лексема	Тип лексемы	Значение
begin	Ключевое слово	A1
if	Ключевое слово	A2
a	Идентификатор	*a*:1
>	Знак операции сравнения	>
b	Идентификатор	*b*:2
then	Ключевое слово	A3
a	Идентификатор	*a*:1
:=	Знак присваивания	:=
a	Идентификатор	*a*:1
-	Знак арифметической операции	-
b	Идентификатор	*b*:2
else	Ключевое слово	A4
a	Идентификатор	*a*:1
:=	Знак присваивания	:=
b	Идентификатор	*b*:2
*	Знак арифметической операции	*
0.3	Вещественная константа	0.3
;	Разделитель	;
end	Ключевое слово	A5
;	Разделитель	;

5.5. Принципы построения лексических анализаторов

Лексический анализатор имеет дело с такими объектами, как различного рода константы и идентификаторы (к последним относятся и ключевые слова).

Язык констант и идентификаторов в большинстве случаев является регулярным, то есть может быть описан с помощью регулярных грамматик.

Распознавателями для регулярных языков являются конечные автоматы (КА).

Существуют правила, с помощью которых для любой регулярной грамматики может быть построен недетерминированный конечный автомат, распознающий цепочки языка, заданного этой грамматикой.

Конечный автомат для каждой входной цепочки языка дает ответ на вопрос о том, принадлежит или нет цепочка языку, заданному автоматом.

5.5. Принципы построения лексических анализаторов

Поскольку во входном тексте программы лексемы не ограничены никакими специальными символами, то выделение границ лексем представляет определенную проблему (в терминах программы-сканера определение границ лексем есть выделение тех строк в общем потоке входных символов, для которых надо выполнять распознавание).

Для большинства входных языков границы лексем распознаются по заданным терминальным символам (пробелы, знаки операций, символы комментариев, разделители (запятые, точки с запятой и т. п.)). Набор таких терминальных символов может варьироваться в зависимости от синтаксиса входного языка.

5.5. Принципы построения лексических анализаторов

Лексический анализатор действует по следующему принципу:

1) Очередной символ из входного потока данных добавляется в лексему всегда, когда он может быть туда добавлен.
2) Как только символ не может быть добавлен в лексему, то считается, что он является границей лексемы и началом следующей лексемы (если символ не является пустым разделителем – пробелом, символом табуляции или перевода строки, знаком комментария).

5.5. Принципы построения лексических анализаторов

Лексический анализатор работает прямо, если для данного входного текста (цепочки символов входного языка) и текущего положения указателя в нем он определяет лексему, расположенную непосредственно справа от указателя, и сдвигает сам указатель вправо от части текста, образующей эту лексему.

При прямой работе ЛА возможно его последовательное взаимодействие с синтаксическим распознавателем

5.5. Принципы построения лексических анализаторов

Лексический анализатор работает **непрямо**, если для данного входного текста (цепочки символов входного языка), заданного типа лексемы и текущего положения указателя в нем он определяет лексему, расположенную непосредственно справа от указателя, и если она соответствует требуемому типу, то сдвигает указатель вправо от части текста, образующей эту лексему.

В отличие от прямого варианта данному ЛА на входе требуется задавать также и тип ожидаемой лексемы. Поэтому при непрямой работе ЛА требуется его параллельное взаимодействие с синтаксическим распознавателем.

5.5. Принципы построения лексических анализаторов

Обобщенный алгоритм работы простейшего ЛА в компиляторе:

1) из входного потока выбирается очередной символ, в зависимости от которого запускается тот или иной ЛА (*символ может быть также проигнорирован либо признан ошибочным*);
2) запущенный ЛА просматривает входной поток символов программы на исходном языке, выделяя символы, входящие в требуемую лексему (*до обнаружения очередного символа, который может ограничивать лексему, либо до обнаружения ошибочного символа*);

5.5. Принципы построения лексических анализаторов

Обобщенный алгоритм работы простейшего ЛА в компиляторе:

3) при успешном распознавании информация о выделенной лексеме заносится в таблицу лексем и ТИ, алгоритм возвращается к первому этапу и продолжает рассматривать входной поток символов с того места, на котором остановился ЛА;
4) при неуспешном распознавании выдается сообщение об ошибке, а дальнейшие действия зависят от реализации ЛА – либо его выполнение прекращается, либо делается попытка распознать следующую лексему (*идет возврат к первому этапу алгоритма*).

5.5. Принципы построения лексических анализаторов

В качестве примера возьмём входной язык, содержащий набор условных операторов ***if … then … else u if … then***, разделённых символом ‘;’ (точка с запятой).

Операторы условия содержат логические выражения, построенные с помощью круглых скобок и операций *or, xor, and*, операндами которых являются идентификаторы и целые десятичные константы без знака.

В исполнительной части эти операторы содержат оператор присваивания (‘:=’) или другой условный оператор.

5.5. Принципы построения лексических анализаторов

Описанный выше входной язык может быть задан с помощью КС грамматики

G ({***if, then, else, ':=', or, xor, and***, '(', ')', ';' , '_', 'a', 'b', 'c', ..., 'x', 'y', 'z',

'A', 'B', 'C', ..., 'X', 'Y', 'Z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ⊥}, {S, F, E, D, C, I, L, N, Z}, P, S)

$S \rightarrow F\text{';'}\perp \mid F\text{';'}S$

$F \rightarrow \textbf{if}\ E\ \textbf{then}\ F\ \textbf{else}\ F \mid \textbf{if}\ E\ \textbf{then}\ F \mid I\ \text{':='}\ E$

$E \rightarrow E\ \textbf{or}\ D \mid E\ \textbf{xor}\ D \mid D$

$D \rightarrow D\ \textbf{and}\ C \mid C$

$C \rightarrow I \mid N \mid \text{'('}E\text{')'}$

$I \rightarrow \text{'\_'} \mid L\ \{\text{'\_'} \mid L \mid \text{'0'} \mid Z\}$

$L \rightarrow \text{'a'} \mid \text{'b'} \mid \text{'c'} \mid \ldots \mid \text{'x'} \mid \text{'y'} \mid \text{'z'} \mid \text{'A'} \mid \text{'B'} \mid \text{'C'} \mid \ldots \mid \text{'X'} \mid \text{'Y'} \mid \text{'Z'}$

$N \rightarrow Z\{\ \text{'0'} \mid Z\}$

$Z \rightarrow \text{'1'} \mid \text{'2'} \mid \text{'3'} \mid \text{'4'} \mid \text{'5'} \mid \text{'6'} \mid \text{'7'} \mid \text{'8'} \mid \text{'9'}$

5.5. Принципы построения лексических анализаторов

Лексемы входного языка разделим на несколько классов:

- шесть ключевых слов языка (***if, then, else, or, xor, and***) – класс 1;
- разделители (‘(‘, ‘)’, ‘;’) – класс 2;
- знак операции присваивания (‘:=’) – класс 3;
- идентификаторы – класс 4;
- целые десятичные константы без знака – класс 5.

Внутреннее представление лексем – это пара вида: *номер_класса*, *номер_в_классе*.

Номер_в_классе – это номер строки в таблице лексем соответствующего класса.

5.5. Принципы построения лексических анализаторов

Границами лексем будут служить пробелы, знаки табуляции, знаки перевода строки и возврата каретки, круглые скобки, точка с запятой и знак двоеточия.

При этом круглые скобки и точка с запятой сами являются лексемами, а знак двоеточия, являясь границей лексемы, в то же время является и началом другой лексемы – операции присваивания.

5.5. Принципы построения лексических анализаторов

Введём следующие переменные:

1) *buf* – буфер для накопления символов лексемы;

2) *c* – очередной входной символ;

3) *d* – переменная для формирования числового значения константы;

4) *TW* – таблица ключевых слов входного языка;

5) *TD* – таблица разделителей входного языка;

6) *TID* – таблица идентификаторов анализируемой программы;

7) *TNUM* – таблица чисел-констант, используемых в программе.

Таблицы *TW* и *TD* заполнены заранее, т.к. их содержимое не зависит от исходной программы;

TID и *TNUM* будут формироваться в процессе анализа; для простоты будем считать, что все таблицы одного типа; пусть *tab* – имя типа этих таблиц, *ptab* – указатель на *tab*.

5.5. Принципы построения лексических анализаторов

Диаграмма состояний для лексического анализатора приведена на рисунке 2. Символом *Nx* на диаграмме обозначен номер лексемы *x* в её классе

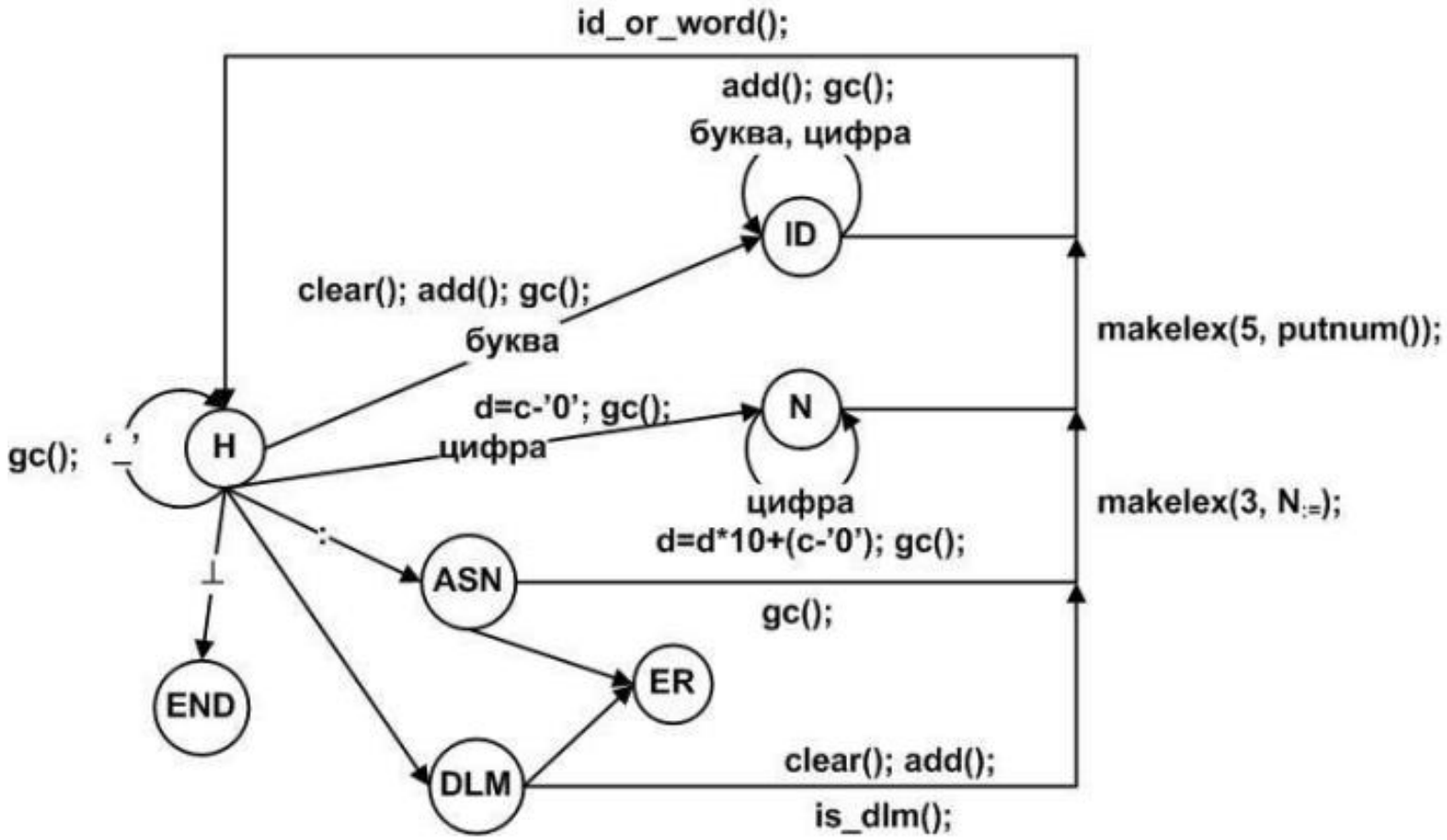

Рисунок 2. Диаграмма состояний для ЛА.

5.5. Принципы построения лексических анализаторов

Функции, используемые лексическим анализатором:

1) *void clear (void);* – очистка буфера *buf*;

2) *void add (void);* – добавление символа с в конец буфера *buf*;

3) *int look (ptab T);* – поиск в таблице T лексемы из буфера *buf.* Функция возвращает номер строки таблицы с информацией о лексеме либо 0, если такой лексемы в таблице T нет;

4) *int putl (ptab T);* – запись в таблицу T лексемы из буфера *buf,* если её там не было. Функция возвращает номер строки таблицы с информацией о лексеме;

5) *int putnum ();* – запись в *TNUM* константы из *d*, если её там не было. Функция возвращает номер строки таблицы *TNUM* с информацией о константе-лексеме;

5.5. Принципы построения лексических анализаторов

Функции, используемые лексическим анализатором:

6) *void makelex (int k, int i);* – формирование и вывод внутреннего представления лексемы; *k* – номер класса, *i* – номер в классе;

7) *void gc (void)*; – функция, читающая из входного потока очередной символ исходной программы и заносящая его в переменную *c*;

8) *void id_or_word (void);* – функция, определяющая является ли слово в буфере *buf* идентификатором или ключевым словом и формирующая лексему соответствующего класса;

9) *void is_dlm (void);* – если символ в буфере *buf* является разделителем, то формирует соответствующую лексему, иначе производится переход в состояние *ER*.

5.6. Программная реализация лексического анализатора

Листинг 1. Лексический анализатор

```
include <stdio.h>
#include <ctype.h>
#define BUFSIZE 80
extern ptab TW, TID, TD, TNUM;
char buf[BUFSIZE]; /* для накопления символов лексемы */
int c; /* очередной символ */
int d; /* для формирования числового значения константы */
int j; /* номер строки в таблице, где находится лексема,
 найденная функцией look */
enum state {H, ID, NUM, ASN, DLM, ER, END};
enum state TC; /* текущее состояние */
FILE* fp;
```

5.6. Программная реализация лексического анализатора

Листинг 1. Лексический анализатор

```
void clear(void); /* очистка буфера buf */
void add(void); /* добавление символа с в конец буфера buf */
int look(ptab); /* поиск в таблице лексемы из buf;
 результат: номер строки таблицы либо 0 */
int putl(ptab); /* запись в таблицу лексемы из buf, если ее
 там не было; результат: номер строки
таблицы */
```

5.6. Программная реализация лексического анализатора

Листинг 1. Лексический анализатор

```
int putnum(); /* запись в TNUM константы из d, если ее там
 не было; результат: номер строки таблицы TNUM */
void makelex(int,int); /* формирование и вывод внутреннего
 представления лексемы */
void id_or_word(void)
{
 if (j=look(TW)) makelex(1,j);
 else
 {
 j=putl(TID); makelex(4,j);
 }
}
```

5.6. Программная реализация лексического анализатора

Листинг 1. Лексический анализатор

```
void is_dlm(void)
{
 if(j=look(TD))
 {
 makelex(2,j);
 gc();
 TC=H;
 }
 TC=ER;
}
```

5.6. Программная реализация лексического анализатора

Листинг 1. Лексический анализатор

```
void gc(void)
{
 c = fgetc(fp);
}
void scan (void)
{
 TC = H;
 fp = fopen("prog","r"); /* в файле "prog" находится текст исходной программы */
 gc();
 do
 {
 switch (TC)
 {
```

5.6. Программная реализация лексического анализатора

Листинг 1. Лексический анализатор

```
case H:
 if (c == ' ') gc();
 else if (isalpha(c))
 {
 clear();
 add();
 gc();
 TC = ID;
 }
```

5.6. Программная реализация лексического анализатора

Листинг 1. Лексический анализатор

```
else if (isdigit (c))
 {
 d = c - '0';
 gc();
 TC = NUM;
 }
else if (c == ':')
 {
 gc();
 TC = ASN;
 }
```

5.6. Программная реализация лексического анализатора

Листинг 1. Лексический анализатор

```
else if (c == '⊥')
 {
 makelex(2, N⊥);
 TC = END;
 }
 else TC = DLM;
 break;
```

5.6. Программная реализация лексического анализатора

Листинг 1. Лексический анализатор

```
case ID:
 if (isalpha(c) || isdigit(c))
 {
 add();
 gc();
 }
 else
 {
 id_or_word();
```

5.6. Программная реализация лексического анализатора

Листинг 1. Лексический анализатор

```
TC = H;
 }
 break;
 case NUM:
 if (isdigit(c))
 {
 d=d*10+(c - '0');
 gc();
 }
```

5.6. Программная реализация лексического анализатора

Листинг 1. Лексический анализатор

```
else
 {
 makelex (5, putnum());
 TC = H;
 }
 break;
 /* ..... */
 } /* конец switch */
 } /* конец тела цикла */
 while (TC != END && TC != ER);
 if (TC == ER) printf("ERROR !!!\n");
 else printf("O.K.!!!\n");
}
```