ЛЕКЦИЯ 5
“ПОСТРОЕНИЕ ЛЕКСИЧЕСКОГО АНАЛИЗАТОРА (СКАНЕРА) НА БАЗЕ КОНЕЧНОГО АВТОМАТА”

ПЛАН

1. Сканер как неотъемлемая часть компилятора
2. Процедура построения лексического анализатора (сканера) на базе конечного автомата.
3. Демонстрационный пример.
4. Синтаксические машины Глени.

Лексический анализатор (сканера)

Сканер – составная часть транслятора, производящая лексический анализ входной (исходной программы).

Лексический анализ программы – анализ синтаксиса на уровне лексических единиц или синтаксический анализ на уровне лексических единиц.

От английского scan [skæn] –глагол, обозначающий внимательно рассматривать, изучать.

В задачи сканера входят.

1. ***Из цепочек литер*** исходной программы ***выделить*** (распознать, составить) символы языка программирования, которые называются ***атомами*** или ***лексемами*** языка.

2. ***Заменить последовательность литер*** переменной длины ***кодами фиксированной разрядности***, соответствующим опознанным лексемам.

3. ***Готовить и сопровождать таблицы соответствия*** объектов исходной программы: идентификаторов и констант системе кодирования лексем, номеров строк программы для облегчения диагностики.

4. ***Представить*** исходную ***программу*** в виде последовательности ***кодов лексем (дескрипторов)***.

Считается, что лексема (или дескриптор лексемы) состоит из двух частей: ***класса*** и ***значения***.

Код класса
Код значения или ссылка

Процедура построение сканера

Сканер строится на основе теории интерпретирующих автоматов. Построение осуществляется в следующей традиционной последовательности.

1. ***Определяется список лексем***.

2. ***Определяются внутренние коды*** лексем. Это могут быть как коды, используемые процессором, для которого создаётся транслятор, так и коды, произвольно назначенные программистом.

3. ***Для лексем*** анализируемого языка необходимо ***построить описание*** пригодное ***для построения конечного автомата***.

4. ***Строится обобщённая диаграмма*** состояний сканера.

5. ***Разрабатываются алгоритмы обработки*** и процедуры их реализации, которые будут вызываться из различных состояний конечного автомата или при переходах между ними. Указанная информация наносится на обобщённую диаграмму.

6. ***Разрабатываются тесты*** для проверки корректности функционирования отдельных модулей и конечного автомата в целом. Определяются ***диагностические сообщения*** об ошибках.

7. ***Составляются спецификации*** программных модулей, входящих в сканер, и ***осуществляется кодирование*** алгоритмов на выбранном или заданном языке программирования.

Пример.

Предложения языка учебного микропроцессора описывается формальной грамматикой. *G*[*Stmt*]:

<Stmt>	::=	***<Lbl> <Cop> <Operand>*** | ***<Dir> <Operand>*** | ***<Cop> <Opernd>*** | ***<Lbl> <Cop>*** | ***<Cop>*** | ***<Lbl> <Dir> <Operand>*** | ***<Dir>***
<Cop>	::=	*add*|*neg*|*jmp*|*shift*
<Dir>	::=	*start*|*org*|*end*
<Operand>	::=	#<*data*>|@<*iden*>|@#<*data*>
<*iden*>	::=	***<Bukva> <Next>*** | ***<Bukva>***
<Bukva>	::=	*a*|*b*|*c*|**…**|*z*
<Next>	::=	***<Bukva> <Next>*** | ***<Cifra> <Next>*** | ***<Bukva>*** | ***<Cifra>***
<Cifra>	::=	*0*|*1*|*2*|**…**|*9*
<*data*>	::=	*0*<***Bin***>*b*|*0*<***Hex***>*h*
<Bin>	::=	***<Bn> <dig2>*** | ***<dig2>***
<dig2>	::=	*0*|*1*
<Hex>	::=	***<Hex> <dig16>*** | ***<dig16>***
<dig16>	::=	*0*|*1*|*2*|**…**|*9*|*a*|*b*|*c*|*d*|*e*|*f*
<Lbl>	::=	<*iden*>

Список лексем и их коды:

Служебные слова			
Операции		***Директивы***	
add	101	*start*	201
neg	102	*org*	202
jmp	103	*end*	203
shift	104		
Разделители			
Однолитерные		***Двулитерные***	
#	301	@#	401
@	302		
Условно-терминальные символы			
Константа		***Идентификатор***	
<*data*>	501	<*iden*>	601

Общая структура регулярного выражения примерно такова:

$$Cc_1L_1 \cup Cc_2L_1 \cup \ldots \cup Cc_iL_1 \cup \ldots \cup Cc_rL_1 \cup <iden>L_1 \cup <data>L_2 \cup d_1 \cup d_2 \cup \ldots \cup d_j \cup \ldots \cup d_n \cup L_3.$$

В записи обозначены компоненты:

Cc_i – регулярное выражение для описания i-того служебного слова;

<*iden*> – регулярное выражение для описания классов переменных и меток;

<*data*> – регулярное выражение для описания класса констант;

L_1 – лексема, состоящая в том, что текущая литера не буква или не цифра;

L_2 – текущая литера не цифра;

d_j – разделители j-того типа;

L_3 – литера, не принадлежащая алфавиту конечного автомата.

Переменную <*iden*> в эскизном виде можно представить регулярным выражением

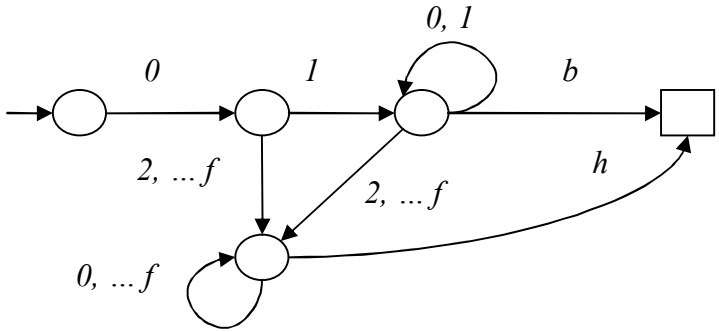
$$Б\ \{Б \vee Ц\},$$

где Б и Ц обозначают букву латинского алфавита и арабскую цифру соответственно.

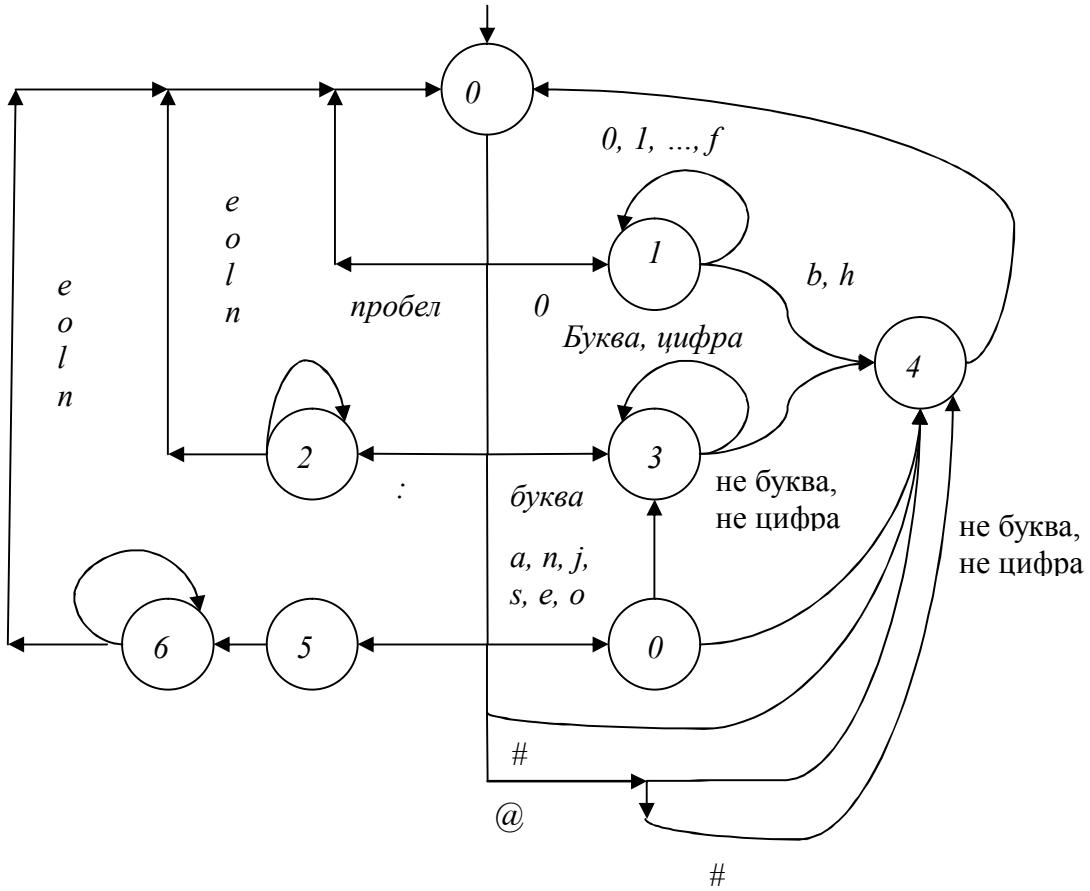
Арифметические константы <*data*> описываются так

$$0\{0 \vee 1\}b \vee 0\{0 \vee 1 \vee 2 \vee 3 \vee 4 \vee 5 \vee 6 \vee 7 \vee 8 \vee 9 \vee a \vee b \vee c \vee d \vee e \vee f\}h.$$

Фрагмент КА для анализа чисел

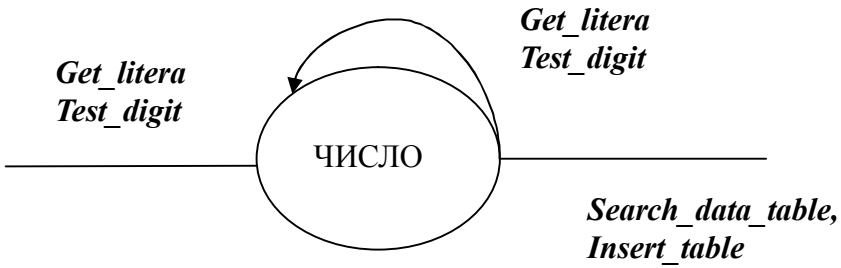


Общая схема сканера



Обозначено: 0 – начальное состояние; 1 – распознавание константы; 2 – игнорирование комментария; 3 – распознавание идентификатора; 4 – формирование лексем и ведение таблиц; 5 – ошибка, встречен символ, который не принадлежит алфавиту; 6 – игнорирование оператора, в котором была допущена ошибка; 7 – распознавание служебных слов.

Нанесение на диаграмму процедур поддержки и обработки приведёт примерно к следующей картине для каждого состояния конечного автомата



Предполагается разработка следующих процедур: ***Get_litera*** – чтение литеры из потока; ***Test_digit*** – проверка на принадлежность литеры к классу цифр; ***Search_data_table*** – определение, является ли сформированный символ новым или уже присутствует в таблице; ***Insert_Table*** – запись нового символа в таблицу.

РАЗРАБОТКА ТЕСТОВОГО ПРИМЕРА

На вход сканера поступает программа на учебном ассемблере.

```
        org     @#030eh ; стартовый адрес
        start
        …
m1      add     @alpha
        …
        end
```

В результате ожидается, что будут построены таблицы констант и идентификаторов

Таблица констант

№	Константа	Значение
1	30e	0000 0011 0000 1110
…	…	…
42	157	0001 0101 0111

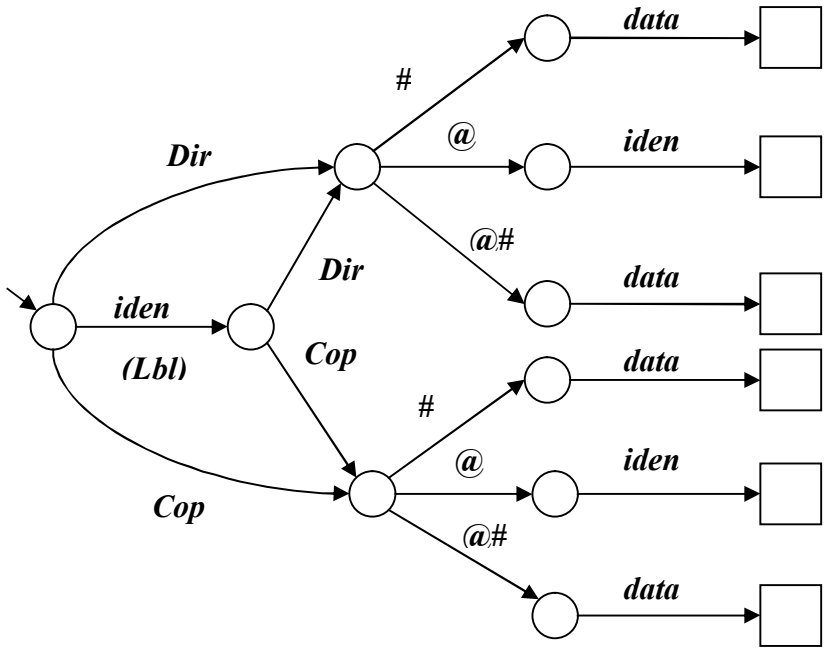
Таблица имён

№	Имя переменной
1	x
…	…
27	alpha
…	…
48	m1

После перекодировки должна получиться следующая последовательность лексем

PROG.COD	202	401	501	…	601	101	302	601	…	203
PROG.NUM	000	000	001	…	048	000	000	049	…	000
ЛЕКСЕМА	`org`	`@#`	`030eh`		`m1`	`add`	`@`	`alpha`		`end`

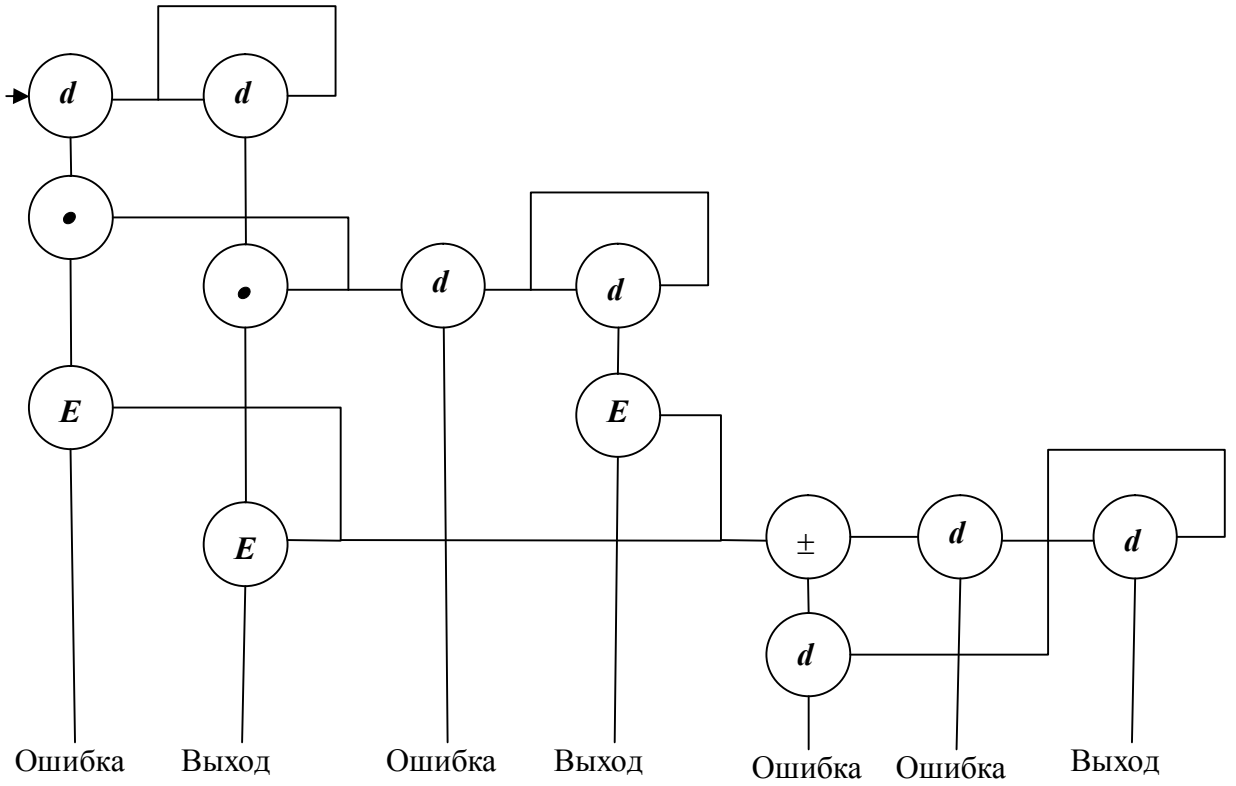
КОНЕЧНЫЙ АВТОМАТ ДЛЯ СИНТАКСИЧЕСКОГО АНАЛИЗА



СИНТАКСИЧЕСКИЕ МАШИНЫ ГЛЕННИ

Glennie A.E. в 1960 г. Кружком обозначено сравнение, выход направо – положительный результат сравнения, при этом входная строка продвигается на одну позицию (1 литеру), выход вниз – отрицательный, продвижение входной строки блокируется

Неминимальная машина Гленни



Машина Гленни с минимальным числом состояний

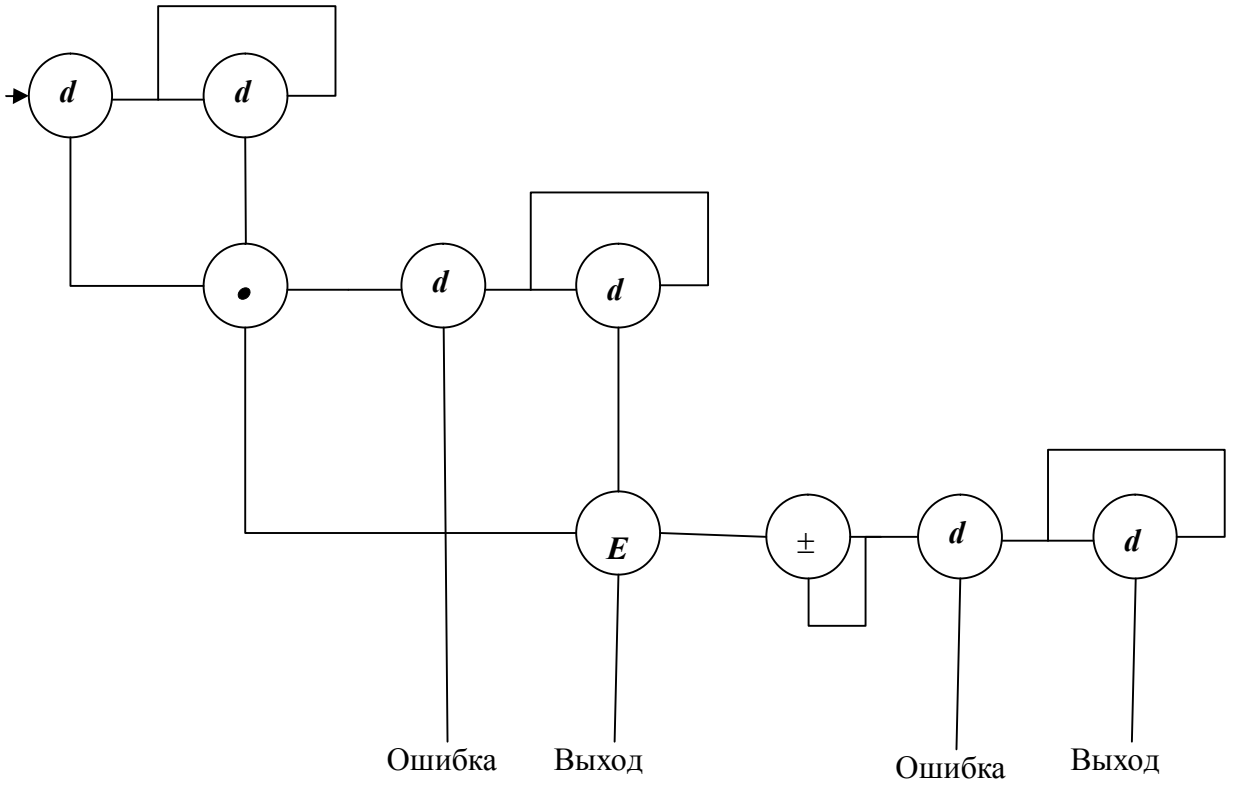


Таблица управления минимальной синтаксической машины

Номер состояния	Символ на входе	Следующее состояние		Выполняемая операция	
		True	False	True	False
1	***d***	2	3	V	–
2	***d***	2	3	V	–
3	**.**	4	6	–	–
4	***d***	5	Error	W	Error
5	***d***	5	6	W	–
6	***E***	7	Exit	–	Y
7	**+** ∨ **-**	8	8	Z	–
8	***d***	9	Error	X	Error
9	***d***		Exit	X	Y

Фрагменты алгоритмов процедур обработки

```
    Procedure V /* подсчёт мантиссы целой части числа при вводе
    /* сдвиг организован умножением на 10 */
    Mantissa := 10 * Mantissa + d
    End V
Procedure W /* подсчёт мантиссы дробной части числа при вводе */
N := N +1 /* подсчёт длины дробной части */
```

```
Mantissa := 10 * Mantissa + d
End W
    Procedure X
    /* подсчёт показателя степени в научной нотации числа */
    Expo :=10 * Expo + d
    End X
Procedure Y
/* окончательное получение эквивалента числа в формате с плавающей
точкой */
Result := Mantissa * 10 ^ (Znak * Expo – N)
End Y
    Procedure Z
    /* определение знака в показателе степени числа в научной
    нотации */
    Znak := sign
    End Z
Procedure Error /* выдача диагностического сообщения */
output (“Ошибка в записи числа”)
End Error
```

Перед началом работы переменные *Mantissa*, *N*, *Expo* положены равными нулю, а переменная *Znak* равна единице, что соответствует знаку плюсу в показателе степени.