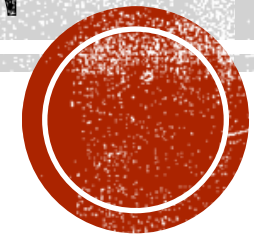
ЯЗЫКИ ПРОГРАММИРОВАНИЯ И МЕТОДЫ ТРАНСЛЯЦИИ



Лекция 4. Построение лексического анализатора

26 сентября 2025 г.

ПРАВИЛА

1. Распознаем самую длинную лексему
2. Ключевые слова считываются как идентификаторы
3. После символа операции не может сразу идти символ другой операции
4. При переходе в ошибочное состояние печатаем сообщение о возникшей лексической ошибке
5. Если автомат заканчивает не в заключительном состоянии, печатаем сообщение об ошибке
6. Лексический анализ проводим по диаграмме переходов

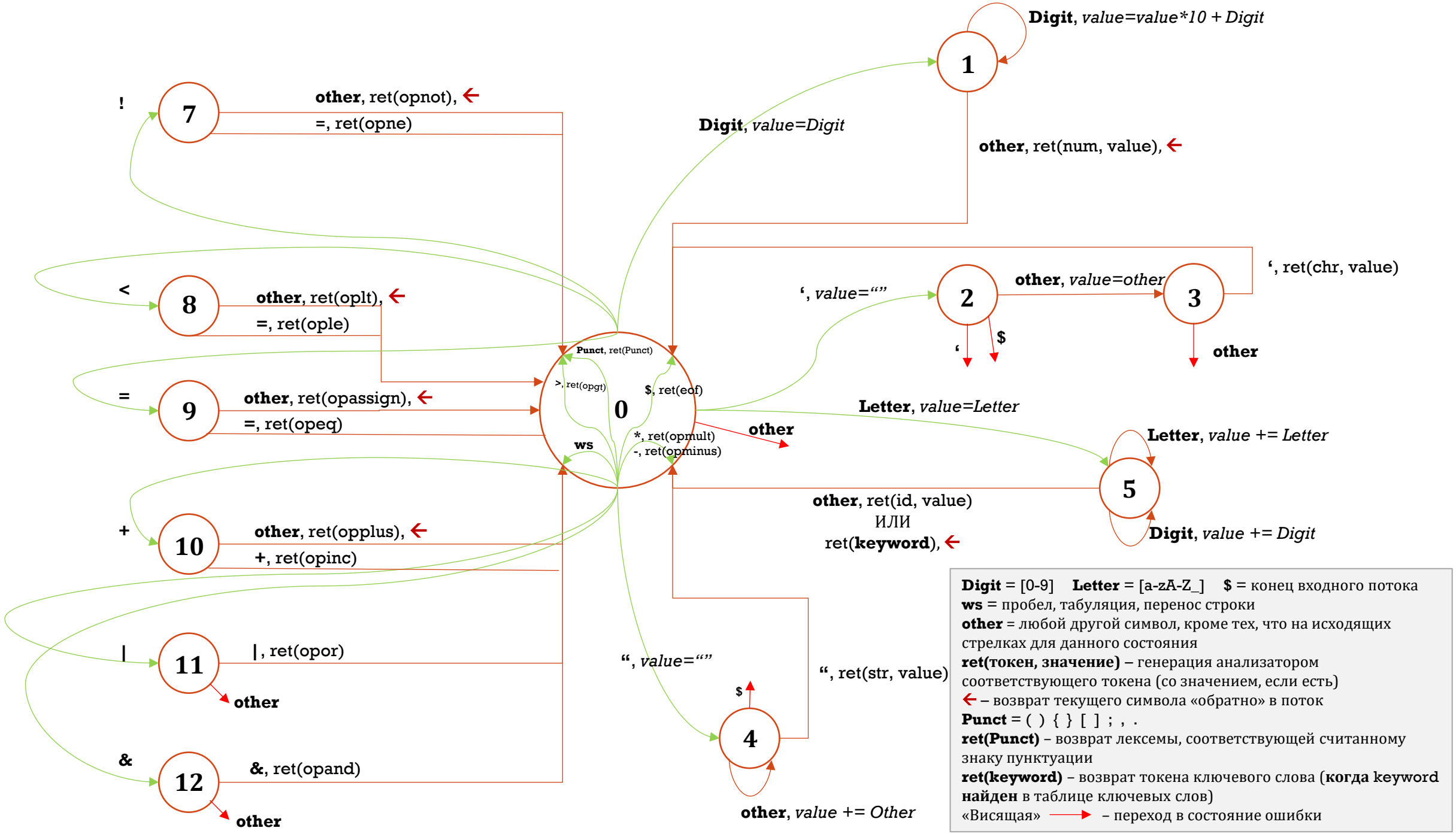
Таблица лексем языка MiniC

№	Лексемы	Кодовое имя	Описание
Лексемы со значением			
1	121, -1, …	num	Числовая константа
2	‘a’, …	chr	Символьная константа
3	“Hello, world!”, …	str	Строковая константа
4	i, j, func, …	id	Идентификатор
Скобки и пунктуация			
5	(	lpar	Открывающая круглая скобка
6	)	rpar	Закрывающая круглая скобка
7	{	lbrace	Открывающая фигурная скобка
8	}	rbrace	Закрывающая фигурная скобка
9	[	lbracket	Открывающая квадратная скобка
10	]	rbracket	Закрывающая квадратная скобка
11	;	semicolon	Точка с запятой
12	,	comma	Запятая
13	:	colon	Двоеточие
Символы операций			
14	=	opassign	Операция присваивания
15	+	opplus	Операция сложения
16	-	opminus	Операция вычитания
17	*	opmult	Операция умножения
18	++	opinc	Операция инкремента
19	==	opeq	Операция равно
20	!=	opne	Операция неравно
21	<	oplt	Операция меньше
22	>	opgt	Операция больше
23	<=	ople	Операция меньше или равно
24	!	opnot	Операция логического отрицания
25	||	opor	Операция логического ИЛИ
26	&&	opand	Операция логического И

Ключевые слова			
27	int	kwint	Тип данных int
28	char	kwchar	Тип данных char
29	if	kwif	Ключевое слово if
30	else	kwelse	Ключевое слово else
31	switch	kwswitch	Ключевое слово switch
32	case	kwcase	Ключевое слово case
33	while	kwwhile	Ключевое слово while
34	for	kwfor	Ключевое слово for
35	return	kwreturn	Ключевое слово return
36	in	kwin	Ключевое слово in
37	out	kwout	Ключевое слово out

Условные обозначения

1. Digit – любая цифра [0-9]
2. Letter – любая буква латинского алфавита [A-Za-z_]
3. Punct – любой символ скобки или пунктуации
4. $ – конец входного потока символов
5. WS – любой пробельный символ (пробел, табуляция, перенос строки)
6. other – любой другой символ кроме тех, что на исходящих стрелках
7. ret(code, value) – создание токена code (со значением value)
8. ret(Punct) – возврат токена считанного знака пунктуации
9. ret(keyword) – возврат токена ключевого слова (**когда** считанный идентификатор найден в таблице ключевых слов)
10. ← – «возврат» текущего символа «обратно» во входной поток
11. «Висящая» красная стрелка → – переход в состояние ошибки


Digit, value=value*10 + Digit
1
Digit, value=Digit
other, ret(num, value), ←
!
7
other, ret(opnot), ←
=, ret(opne)
<
8
other, ret(oplt), ←
=, ret(ople)
=
9
other, ret(opassign), ←
=, ret(opeq)
+
10
other, ret(opplus), ←
+, ret(opinc)
|
11
|, ret(opor)
other
&
12
&, ret(opand)
other
Punct, ret(Punct)
>, ret(opgt)
$, ret(eof)
0
ws
*, ret(opmult)
-, ret(opminus)
other
‘, value=“”
2
other, value=other
3
‘, ret(chr, value)
‘
$
other
Letter, value=Letter
Letter, value += Letter
5
Digit, value += Digit
other, ret(id, value)
ИЛИ
ret(keyword), ←
“, value=“”
4
$
“, ret(str, value)
other, value += Other
Digit = [0-9] Letter = [a-zA-Z_] $ = конец входного потока
ws = пробел, табуляция, перенос строки
other = любой другой символ, кроме тех, что на исходящих стрелках для данного состояния
ret(токен, значение) – генерация анализатором соответствующего токена (со значением, если есть)
← – возврат текущего символа «обратно» в поток
Punct = ( ) { } [ ] ; , .
ret(Punct) – возврат лексемы, соответствующей считанному знаку пунктуации
ret(keyword) – возврат токена ключевого слова (когда keyword найден в таблице ключевых слов)
«Висящая» → – переход в состояние ошибки

ЛА: ЧТЕНИЕ СИМВОЛОВ ИЗ ВХОДНОГО ПОТОКА

```
def getNextChar(self):
    if self.savedChar != None:
        char = self.savedChar
        self.savedChar = None
        return char
    char = self.file.read(1)
    if not char:
        self.Status = 0
        return None
    else:
        return char

def returnChar(self, char):
    self.savedChar = char
```

ЛА: РАБОТА С КЛЮЧЕВЫМИ СЛОВАМИ

```
from enum import Enum

class LexType(Enum):
    num = 1; chr = 2; str = 3; id = 4; lpar = 5; rpar=6; lbrace=7; rbrace=8;
    lbracket=9; rbracket=10; semicolon=11; comma=12; colon=13; opassign=14;
    opplus=15; opminus=16; opmult=17; opinc=18; opeq=19; opne=20; oplt=21;
    opgt=22; ople=23; opnot=24; opor=25; opand=26; kwint=27; kwchar=28; kwif=29;
    kwelse=30; kwswitch=31; kwcase=32; kwwhile=33; kwfor=34; kwreturn=35; kwin=36;
    kwout=37; eof=38; error=39; kwdefault=40; opdec=41; opreturn=42; opge=43
```

```
class Scanner:
    keywords = {
        'int': LexType.kwint, 'char': LexType.kwchar, 'if': LexType.kwif,
        'else': LexType.kwelse, 'switch':LexType.kwswitch, 'case':LexType.kwcase,
        'while':LexType.kwwhile, 'for':LexType.kwfor, 'return':LexType.kwreturn,
        'in':LexType.kwin, 'out':LexType.kwout, 'default':LexType.kwdefault
    }

    def getIdType(self, id):
        if id in Scanner.keywords: return Scanner.keywords[id]
        else: return LexType.id
```

ЛА: ФРАГМЕНТ GETNEXTLEXEM()

```
def getNextLexem(self):
    while (True):
        char = self.getNextChar()
        if self.State == 0:
            if char == None: return Lexem(LexType.eof)
            if char in Scanner.Digit:
                self.Value = int(char)
                self.State = 1
                continue
            if char == "'":
                self.Value = ""
                self.State = 2
                continue
            if char == '"':
                self.Value = ""
                self.State = 4
                continue
            if char in Scanner.Letter:
                self.Value = char
                self.State = 5
                continue
            if char == '<':
                self.State = 8
                continue
```

ЛА: ФРАГМЕНТ GETNEXTLEXEM()

```
if self.State == 9:
    self.State = 0
    if char == '=':
        return Lexem(LexType.opeq)
    self.returnChar(char)
    return Lexem(LexType.opassign)
if self.State == 10:
    self.State = 0
    if char == '+':
        return Lexem(LexType.opinc)
    self.returnChar(char)
    return Lexem(LexType.opplus)
if self.State == 11:
    if char == '|':
        self.State = 0
        return Lexem(LexType.opor)
    return Lexem(LexType.error, "Incomplete OR operator")
```