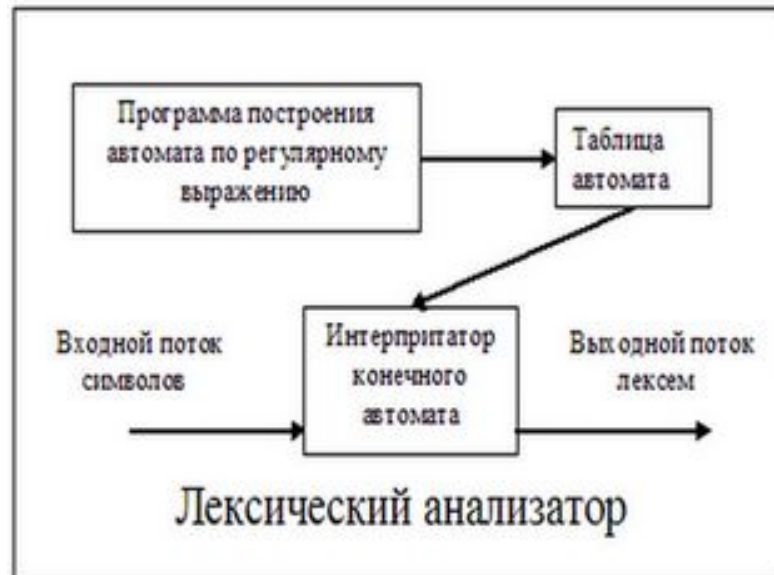
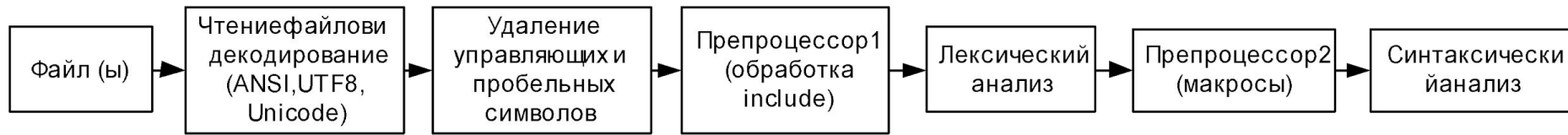
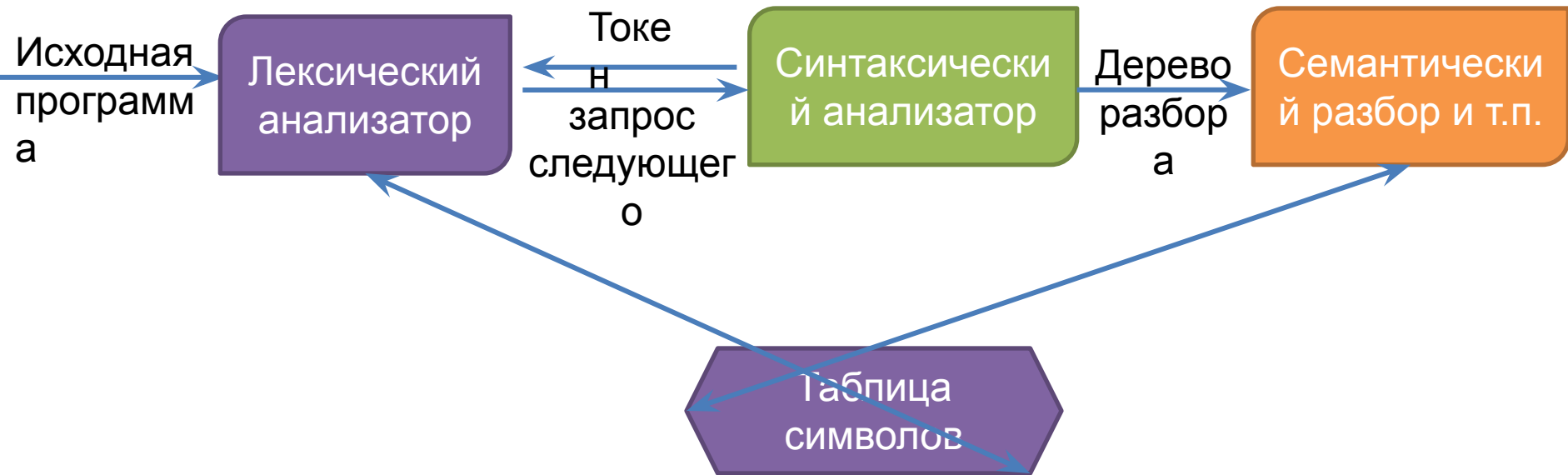


Схема работы лексического анализатора





Токены, шаблоны, лексемы

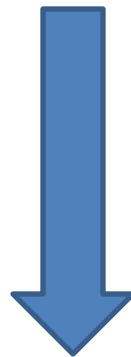
Токен	Пример лексем	Неформальное описание шаблона
const	const	const
if	if	if
relation	<, <=, =, <>, >, >=	< или <= или = или > или >=
id	pi, count, D2	Буква, за которой следуют буквы и цифры
num	3.1416, 0, 6.02E23	Любая числовая константа
literal	"core dumped"	Любые символы между парными кавычками, исклю- исключая сами кавычки

Пример результирующего набора лексем

- 1. Лексема **id** («Идентификатор») с номером лексемы и таблицу с именами идентификаторов. Можно также считать, что выдаётся лексема **id** («Идентификатор») с атрибутом «Имя».
- 2. Лексема **ots** (операция типа сложения) и её подтип – атрибут (+, –, xor, or или иное).
- 3. Лексема **otm** (операция типа умножения) и её подтип – атрибут (*, /, and или иное).
- 4. Лексема **rel** (операция типа сравнения) и её подтип – атрибут (>, <, =, is, as или иное).
- 5. Лексема **lb** (левая скобка)
- 6. Лексема **rb** (правая скобка)
- 7. Лексема **sem** (точка с запятой)
- 8. Лексема **ass** (:=)
- 9. Различные ключевые слова:
 - Лексема **if**.
 - Лексема **then**.
 - Лексема **begin**.
 - ...

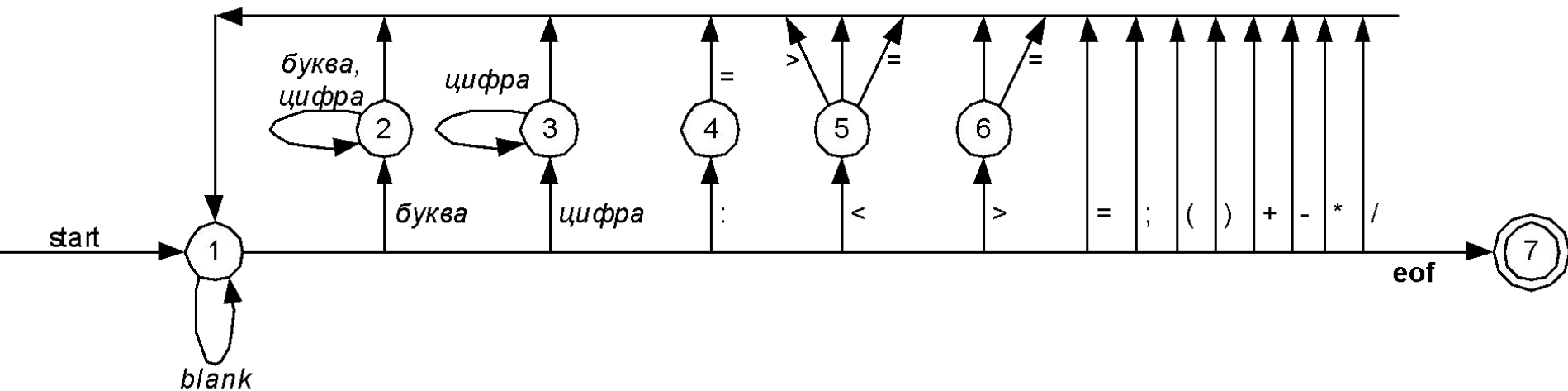
Пример обработки входного потока

```
if distance >= speed * (endtime-starttime) then  
  distance := distance - delta;
```



```
if id rel id otm lb id ots id rb then id ass id ots id  
sem
```

Простейший вариант конечного автомата



Атрибуты лексем

- Атрибуты – это способ хранения дополнительной информации о лексеме (например, конкретное значение константы, имя идентификатора и т.д.)
- Достаточно часто атрибуты представляют собой ссылки в специальные таблицы, например, таблицу представлений, таблицу идентификаторов и т.п.

Пример:

```
bool c; int a=1,b=2;  
c = a>b>>2;
```

Последний оператор порождает следующую последовательность лексем и их атрибутов:

<Identifier_LC, указатель в таблицу на *c*>

<AssignOP_LC, >

<Identifier_LC, указатель в таблицу на *a*>

<GreaterThanOP_LC, >

<Identifier_LC, указатель в таблицу на *b*>

<RightShiftOP_LC, >

<Number_LC, указатель в таблицу на *2*>

Таблица представлений

Место хранения экземпляров (по одному) всех *внешних представлений* идентификаторов (и, возможно, также для всех констант). Позже идентификаторы заменяются на ссылку в эту таблицу – этот процесс называется *свертыванием*.

- Простейший вид таблицы представлений – массив указателей на строки
- Требуется возможность расширения и быстрого поиска
- Более распространенная форма организации таблицы представлений – в виде набора хэшированных списков

Разработка лексера с Coco/R

- Coco/R использует метод рекурсивного спуска для анализа LL(1)-грамматик, т.е. грамматик, для которых можно по 1 следующему символу сразу понять, какое правило было применено для порождения цепочки.
- **Пример.** Не LL(1)-грамматика:
Statement ::= Ident ':=' Expression | Ident ['(' ExpressionList ')']
- **Пример.** LL(1)-грамматика:
Statement ::= Ident (':=' Expression | ['(' ExpressionList ')'])
-
- Однако, используя некоторые дополнительные возможности, реально можно разбирать LL(k)-грамматики с $k > 1$.
- В настоящее время есть версии Coco/R для **C, C++, C#, Java, Pascal, Delphi, Modula-2, Oberon, Ada, Ruby, Unicon.**

Грамматика Pascal для Сосо/R

COMPILER Pascal

IGNORE CASE

CHARACTERS

eol = CHR(13) .

letter = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz" .

digit = "0123456789" .

noQuote1 = ANY - "" - eol .

IGNORE CHR(9) .. CHR(13)

COMMENTS FROM "(" TO ")"

COMMENTS FROM "{" TO "}"

TOKENS

identifier = letter { letter | digit } .

integer = digit { digit } | digit { digit } CONTEXT ("..") .

real = digit { digit } "." digit { digit }

["E" ["+" | "-"] digit { digit }]

| digit { digit } "E" ["+" | "-"] digit { digit } .

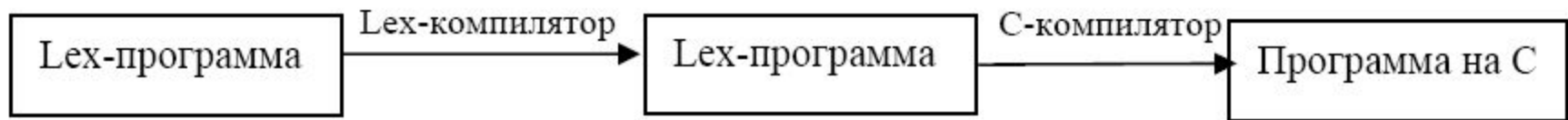
string = "" { noQuote1 | "" } "" .

PRODUCTIONS

...

Разработка лексера с lex/Flex

- Инструмент для создания лексически анализаторов – Lex, состоящий из Lex-языка и Lex-компилятора.



- Lex-программа состоит из трех частей: описаний, правил трансляции и процедур. Каждая часть отделяется от следующей строкой, содержащей два символа %%.

Грамматика Pascal для lex

```
%{  
#include <stdio.h>  
#include "y.tab.h"  
int line_no = 1;  
%}
```

A [aA]

B [bB]

C [cC]

D [dD]

E [eE]

F [fF]

G [gG]

H [hH]

I [iI]

J [jJ]

K [kK]

L [lL]

M [mM]

N [nN]

{A}{N}{D} return(AND);
{A}{R}{R}{A}{Y} return(ARRAY);
{C}{A}{S}{E} return(CASE);
{C}{O}{N}{S}{T} return(CONST);
{D}{I}{V} return(DIV);
{D}{O} return(DO);
{D}{O}{W}{N}{T}{O} return(DOWNTO);
{E}{L}{S}{E} return(ELSE);
{E}{N}{D} return(END);
{E}{X}{T}{E}{R}{N} |
{E}{X}{T}{E}{R}{N}{A}{L} return(EXTERNAL);
{F}{O}{R} return(FOR);
{F}{O}{R}{W}{A}{R}{D} return(FORWARD);
{F}{U}{N}{C}{T}{I}{O}{N} return(FUNCTION);
{G}{O}{T}{O} return(GOTO);
{I}{F} return(IF);
{I}{N} return(IN);
{L}{A}{B}{E}{L} return(LABEL);
{M}{O}{D} return(MOD);
{N}{I}{L} return(NIL);
{N}{O}{T} return(NOT);
{O}{F} return(OF);
{O}{R} return(OR);
{O}{T}{H}{E}{R}{W}{I}{S}{E} return(OTHERWISE);

```
":=" return(ASSIGNMENT);
'({NQUOTE}|")+' return(CHARACTER_STRING);
":" return(COLON);
"," return(COMMA);
[0-9]+ return(DIGSEQ);
"." return(DOT);
".." return(DOTDOT);
"=" return(EQUAL);
">=" return(GE);
">" return(GT);
"[" return(LBRAC);
"<=" return(LE);
"(" return(LPAREN);
"<" return(LT);
"-" return(MINUS);
"<>" return(NOTEQUAL);
"+" return(PLUS);
"]" return(RBRAC);
[0-9]+ "." [0-9]+ return(REALNUMBER);
")" return(RPAREN);
";" return(SEMICOLON);
"/" return(SLASH);
"*" return(STAR);
***" return(STARSTAR);
```

```

"(*" |
"{" { register int c;
    while ((c = input()))
    {
        if (c == '}')
            break;
        else if (c == '*')
        {
            if ((c = input()) == ')')
                break;
            else
                unput (c);
        }
        else if (c == '\n')
            line_no++;
        else if (c == 0)
            commenteof();
    }
}

```

[\t\f] ;

Простейший лексический анализатор

```
while (1)
{ fix=i;
switch(s[i])
{
case "": // Распознавание строковой
константы "... " с двойными ""
внутри
mmm: i++; while (s[i] != "") i++;
i++; if (s[i] == "") goto mmm;
lexem(1); break;
case '/': i++; // Распознавание / и /*
if (s[i] != '*')
{ lexem(14); break; }
// Распознавание комментария /* ... */
n1: while (s[i] != '*') i++;
i++; if (s[i] == '/')
{ i++; lexem(2); break; }
goto n1;
```

```
case '+': i++;
// Распознавание += и +
if (s[i] == '=')
{ i++; lexem(5); }
else lexem(15);
break;
case '<': i++;
// Распознавание << и <
if (s[i] == '<')
{ i++; lexem(6); }
else lexem(16); break;
default: if (isalpha(s[i]))
// Распознавание идентификатора
{ i++; while (isalpha(s[i])) i++;
lexem(11); break; }
if (isdigit(s[i]))
// Распознавание константы
{ i++; while (isdigit(s[i])) i++;
lexem(12); break; }
```

Подсчет числа слов и строк в файле

```
/****** Раздел определений *****/
NODELIM [^" "\t\n]
/* NODELIM означает любой символ, кроме разделителей слов */
int l, w, c; /* Число строк, слов, символов */
%% /****** Раздел правил *****/
{ l=w=c=0; /* Инициализация */ }
{NODELIM}+ { w++; c+=yyleng; /* Слово */ }
\n { l++; /* Перевод строки */ }
. { c++; /* Остальные символы */ }
%% /****** Раздел программ *****/
int main()
{ yylex(); }
yywrap()
{
printf( " Lines - %d Words - %d Chars - %d\n", l, w, c );
return( 1 );
}
```

Специальные конструкции и функции Lex

`ytext` – указатель на отождествленную цепочку символов, оканчивающуюся нулем;

`yleng` – длина этой цепочки

`yless(n)` – вернуть последние `n` символов цепочки обратно во входной поток;

`yymore()` – считать следующие символы в буфер `ytext` после текущей цепочки

`yynput(c)` – поместить байт `c` во входной поток

`ECHO` – копировать текущую цепочку в `yout`

`ylval` – еще одно возвращаемое значение

Заглядывание вперед при лексическом анализе

- В некоторых случаях может потребоваться заглядывание вперед для точного определения текущей лексемы
- Для этого используется выражение A/B , которое выдает лексический класс A только в том случае, если за ним следует B
- $DO \ 5 \ I=1,25$
- $DO / (\{letter\} | \{digit\})^* = (\{letter\} | \{digit\})^*$,