

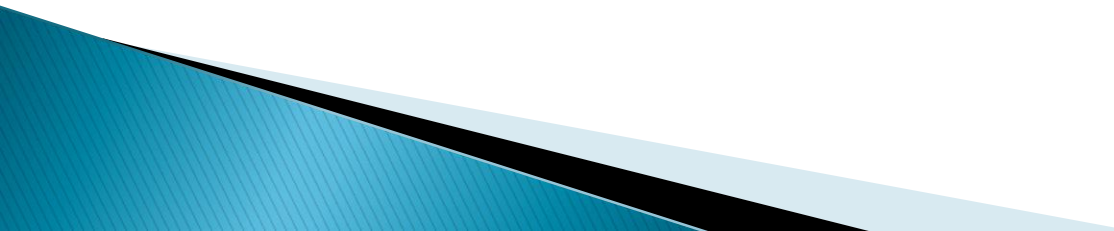
Limbaje formale, automate și compilatoare

Curs 8


Limbaje formale și automate

- ▶ Limbaje de tipul 3
 - Gramatici regulate
 - Automate finite
 - Deterministe
 - Nedeterministe
 - Expresii regulate
 - $a, a \in \Sigma, \epsilon, \emptyset$
 - $E_1.E_2, E_1|E_2, E_1^*, (E_1)$

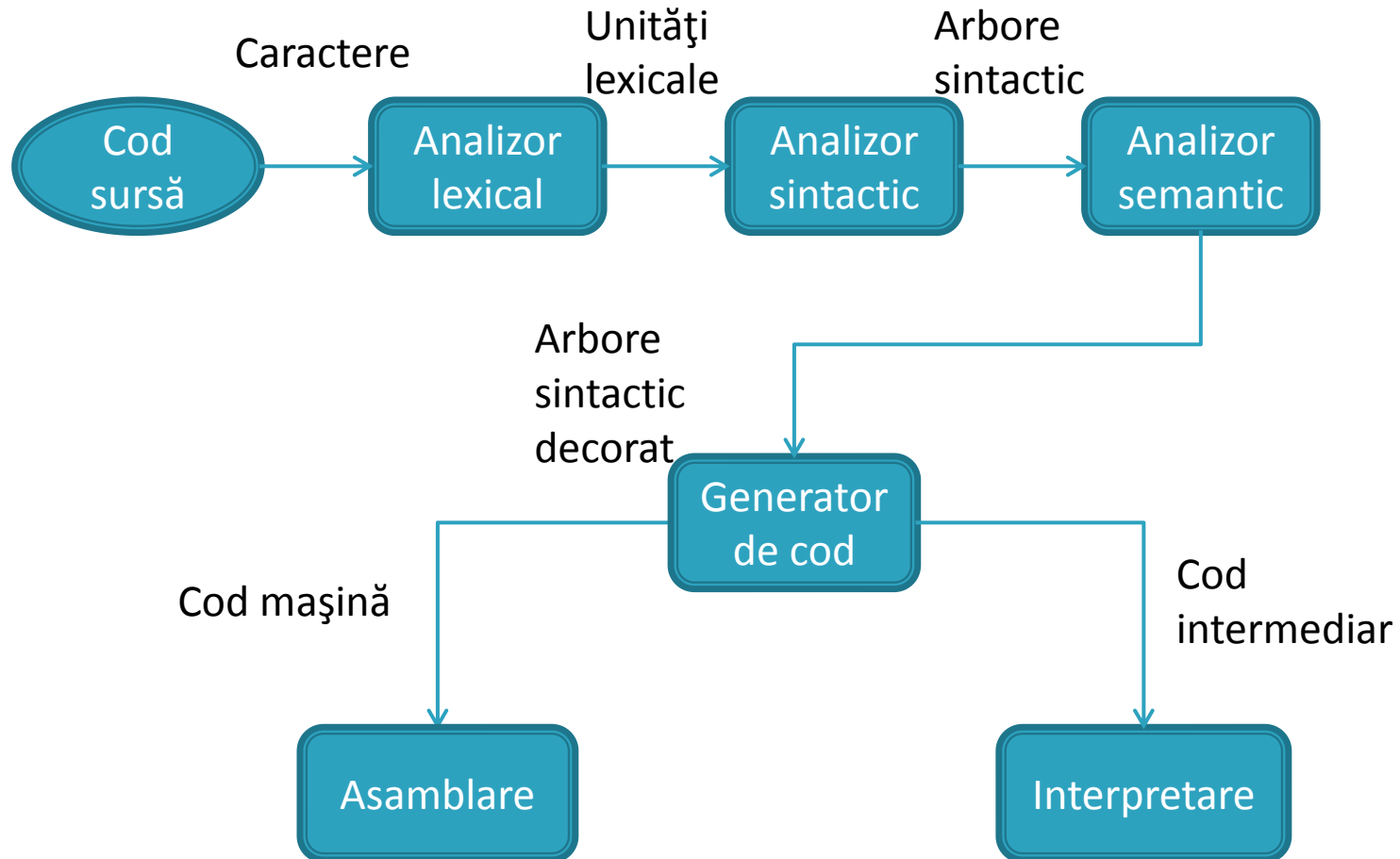
Plan

- ▶ Istoric
 - ▶ Pașii compilării
 - ▶ Analiza lexicală
 - Descriere lexicală
 - Interpretare
 - Interpretare orientată dreapta
 - Descriere lexicală bine formată
 - ▶ Lex
- 

Istoric

- ▶ 1957 Fortran: primul compilator (expresii aritmetice, instrucțiuni, proceduri)
 - ▶ 1960 Algol: prima utilizare a definițiilor formale (gramatici, BNF, structura de bloc, recursie)
 - ▶ 1970 Pascal: tipuri utilizator, mașini virtuale (P-code)
 - ▶ 1972 C: variabilele dinamice, multitasking, gestionarea întreruperilor
 - ▶ 1983 ADA: primul limbaj standardizat
 - ▶ 1985 C++: orientare-obiect, excepții, template-uri
 - ▶ 1995 Java: just-in-time compilation
 - ▶ 2000 C#: Tehnologia .NET
- 

Compilare



Analiza lexicală

- ▶ **Def. 1** Fie Σ un alfabet (al unui limbaj de programare). O *descriere lexicală* peste Σ este o expresie regulată $E = (E_1 | E_2 | \dots | E_n)^+$, unde n este numărul unităților lexicale, iar E_i descrie o unitate lexicală, $1 \leq i \leq n$.
- ▶ **Def. 2** Fie E o descriere lexicală peste Σ ce conține n unități lexicale și $w \in \Sigma^+$. Cuvântul w este *corect relativ la descrierea* E dacă $w \in L(E)$. O *interpretare* a cuvântului $w \in L(E)$ este o secvență de perechi $(u_1, k_1), (u_2, k_2), \dots, (u_m, k_m)$, unde $w = u_1 u_2 \dots u_m$, $u_i \in L(E_{k_i})$ $1 \leq i \leq m$, $1 \leq k_i \leq n$.

Exemplu

- ▶ `w = alpha := beta = 542`
- ▶ Interpretări ale cuvântului `w`:
 - `(alpha, Id), (:=, Asignare), (beta, Id), (=, Egal), (542, Intreg)`
 - `(alp, Id), (ha, Id), (:=, Asignare), (beta, Id), (=, Egal), (542, Intreg)`
 - `(alpha, Id), (:, Dp), (=, Egal), (beta, Id), (=, Egal), (542, Intreg)`

Analiza lexicală

- ▶ **Def. 3** Fie E o descriere lexicală peste Σ și $w \in L(E)$. O interpretare a cuvântului w , $(u_1, k_1)(u_2, k_2), \dots(u_m, k_m)$, este *interpretare drept –orientată* dacă $(\forall i) 1 \leq i \leq m$, are loc:
 $|u_i| = \max\{|v|, v \in L(E_1 | E_2 | \dots | E_n) \cap \text{Pref}(u_i u_{i+1} \dots u_m)\}.$
(unde $\text{Pref}(w)$ este mulțimea prefixelor cuvântului w).
- ▶ Există descrieri lexicale E în care nu orice cuvânt din $L(E)$ admite o interpretare drept–orientată.
- ▶ $E = (a | ab | bc)^+$ și $w = abc$.

Analiza lexicală

- ▶ **Def. 4** O descriere lexicală E este *bine-formată* dacă orice cuvânt w din limbajul $L(E)$ are exact o interpretare drept-orientată.
- ▶ **Teoremă** Dată o descriere lexicală E este decidabil dacă E este bine formată
- ▶ **Def. 5** Fie E o descriere lexicală bine formată peste Σ . Un *analizor lexical (scanner)* pentru E este un program ce recunoaște limbajul $L(E)$ și produce, pentru fiecare $w \in L(E)$, interpretarea sa drept-orientată.

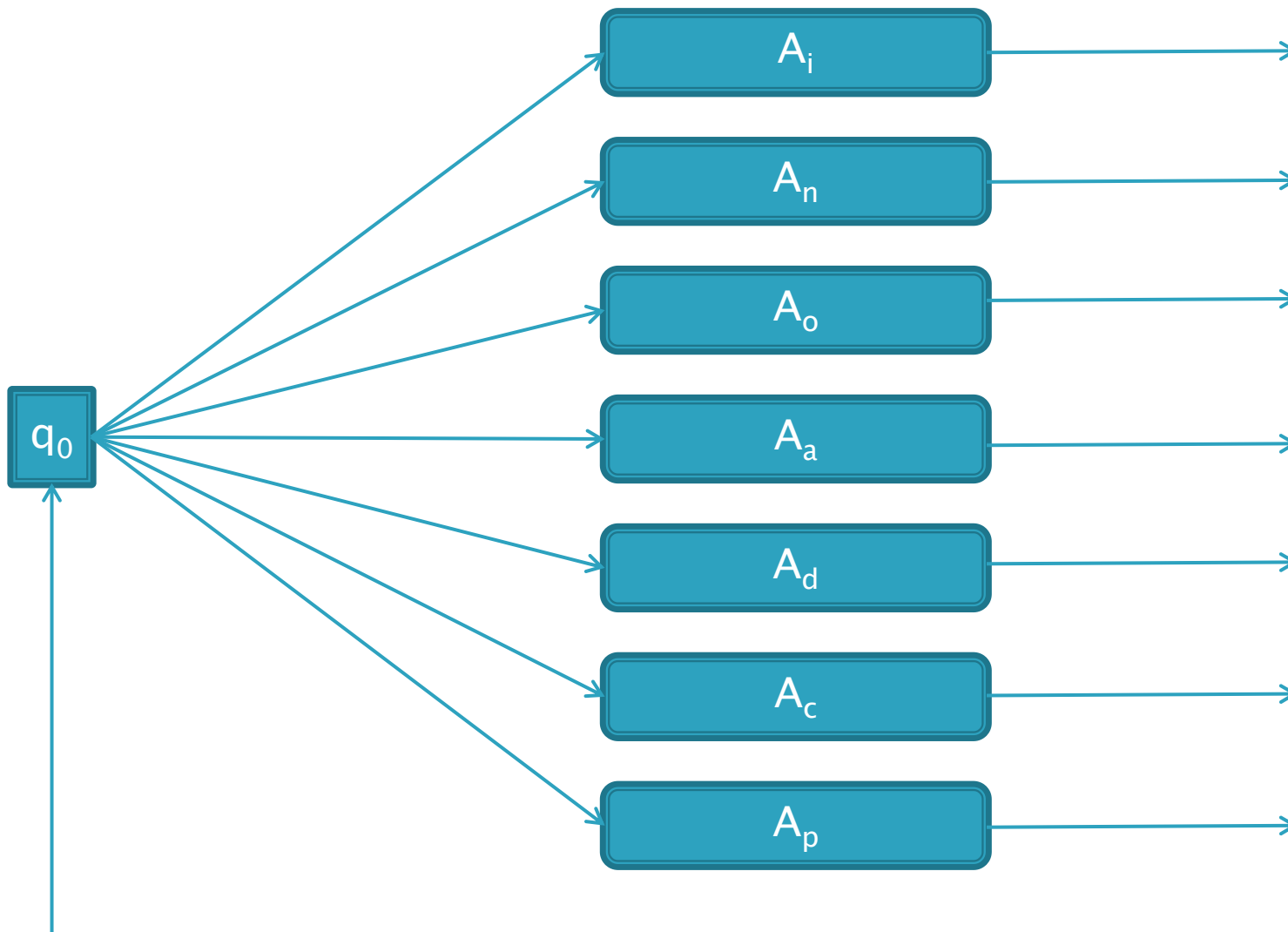
Analiza lexicală

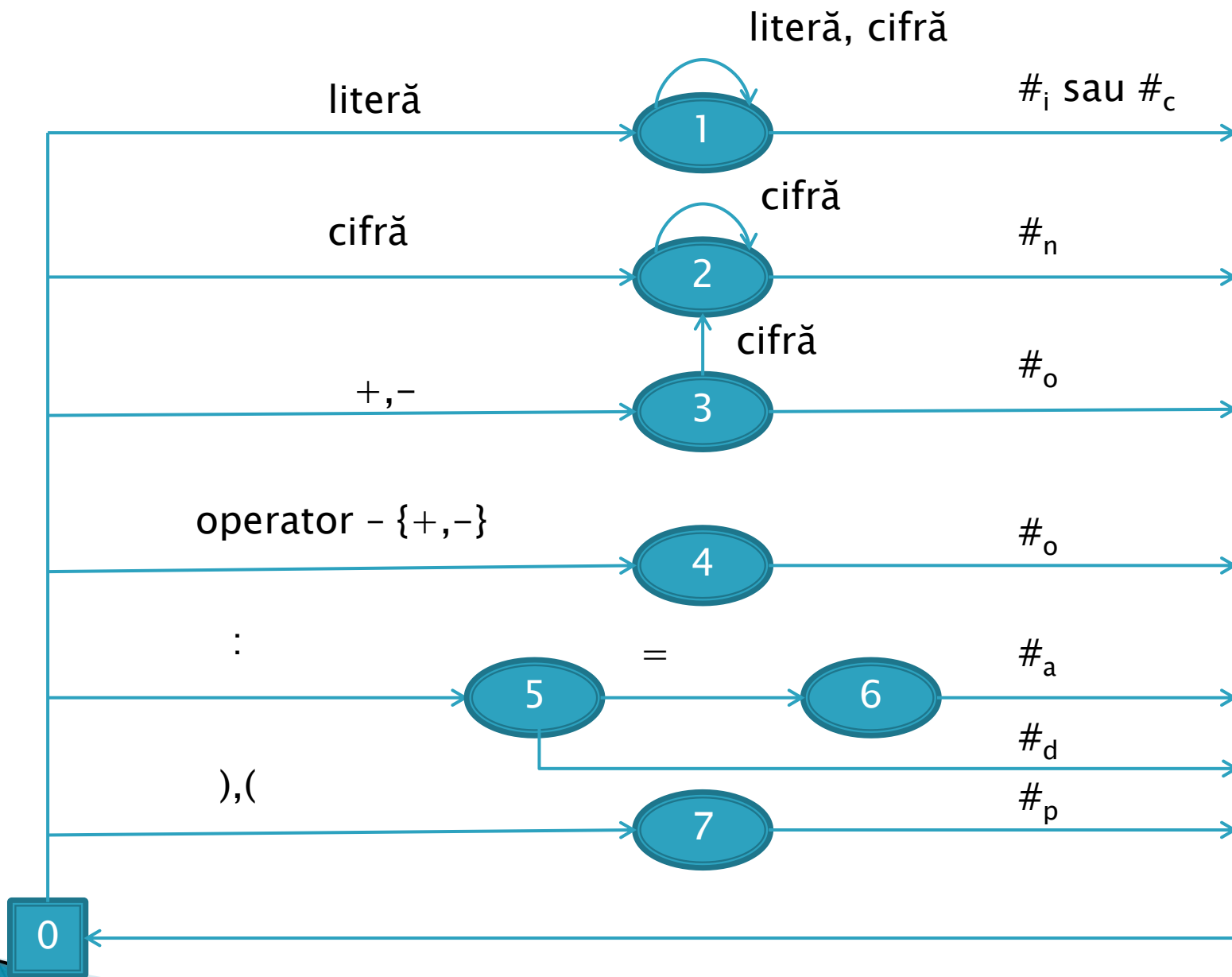
- ▶ Fie o descriere lexicală E peste Σ . Crearea unui analizor lexical pentru E înseamnă:
 - 1. Se construiește automatul finit echivalent A
 - 2. Din A se obține automatul determinist echivalent cu E , fie acesta A' .
 - 3. (Opțional) Automatul minimal echivalent cu A' .
 - 4. Implementarea automatului A' .

Exemplu de analizor lexical

► Fie descrierea lexicală:

- litera $\rightarrow a \mid b \mid \dots \mid z$
- cifra $\rightarrow 0 \mid 1 \mid \dots \mid 9$
- identificator $\rightarrow \text{litera} (\text{litera} \mid \text{cifra})^*$
- semn $\rightarrow + \mid -$
- numar $\rightarrow (\text{semn} \mid \epsilon) \text{cifra}^+$
- operator $\rightarrow + \mid - \mid * \mid / \mid < \mid > \mid <= \mid >= \mid < >$
- asignare $\rightarrow :=$
- doua_puncte $\rightarrow :$
- cuvinte_rezervate $\rightarrow \text{if} \mid \text{then} \mid \text{else}$
- paranteze $\rightarrow) \mid ($

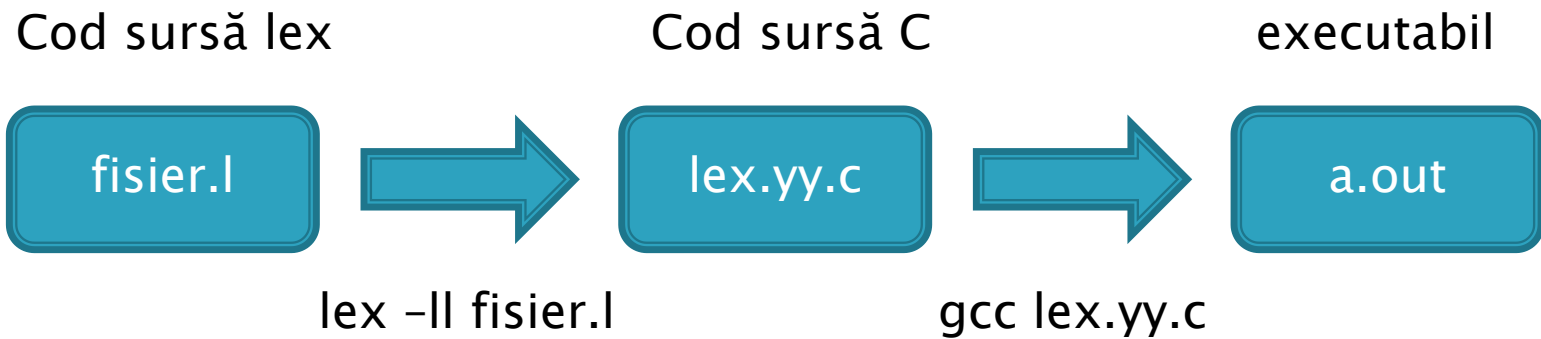




Lex

- ▶ Bell Laboratories 1975 M.E. Lesk și E. Schmidt
- ▶ Standard în UNIX începând cu versiunea a 7-a
- ▶ Variante:
 - FLEX (Fast LEXical Analyzer Generator)
<http://flex.sourceforge.net/>
 - PCLEX lansat de Abraxax Software Inc. (Windows)
 - YooLex (Yet another Object-Oriented Lex)
<http://yoolex.sourceforge.net/>
 - Flex++: <http://www.kohsuke.org/flex++bison++/>
(variantele Bison, Flex care produc cod C++)

Lex – rulare



Lex – sintaxă

- ▶ Trei segmente, separate de %%
 - Declarații
 - Reguli
 - Cod C
- ▶ Declarațiile conțin
 - Declarații C, între secvențele rezervate %{, %}
 - Definiții Lex pentru segmentul de reguli

Lex – sintaxa

- ▶ O definiție Lex are forma
 - `<nume> <expresie_regulată>`
 - Expresiile regulate sunt construite pornind de la orice caracter și folosind operatorii
 - `" \ [] ^ - ? . * + | () / { } % < >`
 - `cifra [0-9]`
 - `litera [a-zA-Z]`

Lex – simboluri rezervate

Simbol	Descriere
.	orice caracter cu excepția newline
\	secvență escape
*	zero sau mai multe copii ale expresiei precedente
+	una sau mai multe copii ale expresiei precedente
?	zero sau o copie a expresiei precedente
^	negație
a b	a sau b
()	grupare de caractere
a+b	literalul "a+b"
[]	clasa de caractere

Lex – expresii

Expresia	Candidați ce se potrivesc
abc	abc
abc*	ab abc abcc abccc ...
abc+	abc abcc abccc ...
a(bc)+	abc abcbc abcbcbc ...
a(bc)?	a abc
[abc]	unul dintre caracterele: a, b, c
[a-z]	orice literă mică
[a\ -z]	unul din caracterele: a, -, z
[-az]	unul din caracterele: -, a, z
[A-Za-z0-9]+	unul sau mai multe caractere alfanumerice
[\t\n]+	spații
[^ab]	orice cu excepția caracterelor: a, b
[a^b]	unul din caracterele : a, ^, b
[a b]	unul din caracterele : a, , b
a b	unul din caracterele : a, b

Lex – sintaxa

▶ Secțiunea de reguli

- exp_1 {Acțiune_1}
- exp_2 {Acțiune_2}
- .
- .
- .
- exp_n {Acțiune_n}

▶ Regulile sunt aplicate în ordinea scrierii

▶ Prima regulă care acceptă cuvântul este aleasă

Lex – elemente predefinite

Nume	Descriere
<code>int yylex(void)</code>	Apelul către analizor
<code>char *yytext</code>	pointer la cuvântul găsit
<code>yylen</code>	lungimea cuvântului găsit
<code>yyval</code>	valoarea asociată cuvântului
<code>FILE *yyout</code>	fișierul de ieșire
<code>FILE *yyin</code>	fișierul de intrare

Exemplul 1

```
%{  
int yylineno;  
%}  
%%  
^(.*)\n printf("%4d\t%s", ++yylineno, yytext);  
%%  
int main(int argc, char *argv[]) {  
    yyin = fopen(argv[1], "r");  
    yylex();  
    fclose(yyin);  
}
```

Exemplul 2

```
letter [A-Za-z]
digit [0-9]
%{
int count;
}%
%%
{letter} ({letter}|{digit}) * {count++;}
.{}
%%
int main(void) {
    yylex();
    printf("number of identifiers = %d\n", count);
    return 0;
}
```

Exemplul 3

```
%{  
int nchar, nword, nline;  
%}  
%%  
\n { nline++; nchar++; }  
[^ \t\n]+ { nword++, nchar += yyleng; }  
. { nchar++; }  
%%  
int main(void) {  
    yylex();  
    printf("%d\t%d\t%d\n", nchar, nword, nline);  
    return 0;  
}
```


Exemplul 4

```
%{
# include <stdio.h>
%}
litera [a-zA-Z]
cifra [0-9]
cifre ({cifra})+
semn [+ -]
operator [+*/<>= -]
spatiu [' ''\t''\n']
%%
"if" | "then" | "else"
({litera})({litera}|{cifra})*
{cifre}|({semn})({cifre})
{operator}
\:\=
\:
(\(|\(|\))
{spatiu}
.
%%
int main( ){
    yylex( );
    return 0;
}
```

```
{printf("%s cuvant rezervat\n", yytext);}
{printf("%s identificador\n", yytext);}
{printf("%s numar intreg\n", yytext);}
{printf("%c operator\n", yytext[0]);}
{printf("%s asignare\n", yytext);}
{printf("%c doua puncte\n", yytext[0]);}
{printf("%c paranteza\n", yytext[0]);}
{}
{printf("%c caracter ilegal\n", yytext[0]);}
```

Bibliografie

- ▶ Grigoraș Gh., *Construcția compilatoarelor. Algoritmi fundamentali*, Editura Universității “Alexandru Ioan Cuza”, Iași, 2005