LEX / FLEX

LEX -> generator de analizor LEXical

~ 1975 Mike Lesk , Eric Schmidt

FLEX - Fast LEX

~ 1987 Vern Paxson

doc: “*pattern-matching techniques used by flex: deterministic finite automata*” *(DFA)*

Expresii regulare

x	se *potriveste* cu caracterul ***x***
r*	zero sau mai multi ***r***
r+	unul sau mai multi ***r***
r?	zero sau un ***r*** (optional)
rs	concatenarea lui ***r*** cu ***s***
r|s	sau ***r*** sau ***s***
r{2,5}	doi, trei, patru sau cinci ***r***
r{2,}	doi sau mai multi r
r{4}	exact patru ***r***
	"character class"
[xyz]	se potriveste cu oricare dintre 'x', 'y', 'z'.
[abj-oZ]	se potriveste cu oricare dintre 'a', 'b', orice de la 'j' la 'o', sau 'Z'
[^A-Z]	*"negated character class"*
	se potriveste cu orice caracter cu exceptia celor specificate
.	orice caracter cu exceptia newline
{name}	se *potriveste* cu definitia "name"

Exemplu:

```
DIGIT       [0-9]
ID          [a-z][a-z0-9]*

%%

if|then|while|do|begin|end
"+"|"-"|"*"|"/"
[ \t\r\n]                   /*  white spaces */
{DIGIT}+
{DIGIT}+"."{DIGIT}*
{ID}
```

Exemplu:

```
%{
#include <stdio.h>
%}

alpha    [A-Za-z]
digit    [0-9]
alphanum  [A-Za-z0-9]
%%
[ \t\r\n]                  /*  ignore white spaces */
"if"                       printf("if \n");
"then"                     printf("then \n");
"else"                     printf("else \n");
"while"                    printf("while \n");
"do"                       printf("do \n");
"begin"                    printf("begin \n");
"end"                      printf("end \n");
{alpha}{alphanum}*         printf("identifier \n");
{digit}+                   printf("number \n");
.                          /* ignore other character*/
%%
```

Formatul fisierului de specificatii

```
%{
        // text copiat in fisierul generat
        // - incluziuni de fisiere, variabile globale
%}
definitii
%%
reguli
%%
cod utilizator
```

Sectiunea de definitii

declaratii de *definitii de nume* simple, de forma

```
nume definitie
```

- **nume** - este un cuvant compus din una sau mai multe litere, cifre, '_' sau '-'

  primul caracter trebuie sa fie litera sau '_'

  si sa se afle pe prima pozitie a liniei.
- **definitie** este o expresie regulara si se considera ca incepe cu primul caracter non-spatiu de dupa nume si tine pana la sfarsitul liniei.

Scop: pentru a simplifica specificarea regulilor

Sectiunea de reguli

Scopul principal al acestei sectiuni este acela de a asocia

- expresii regulare (consideram ca o expr. regulara descrie un atom)
- actiuni *semantice* (cod C definit de utilizator).

Pentru aceasta se utilizeaza o constructie de forma:

sablon actiune

unde:

- **sablon** este o expresie regulara
- **actiune** este o secventa formata din 1 sau mai multe instructiuni C

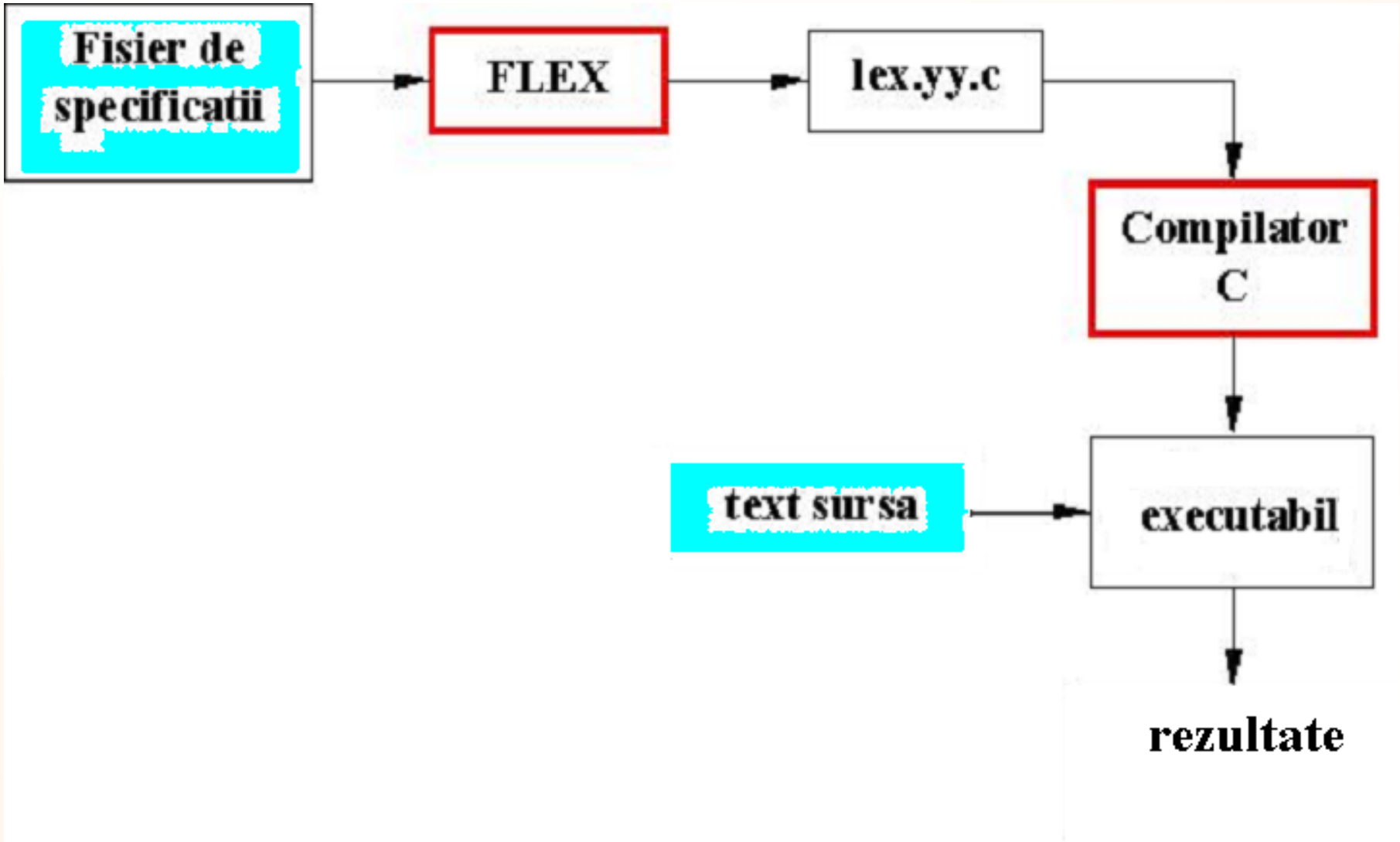
  Acestea trebuie sa inceapa pe aceeasi linie cu sablonul.

  Daca dorim sa scriem instructiuni pe mai multe linii,

  acestea se vor inchide intre acolade.

  In particular, actiunea poate fi si vida.

Cum se lucreaza cu FLEX



Executie analizor generat cu flex

Lansarea în execuţie flex

flex [*optiuni*] [*nume_fisier_specificatii*]

- unde *nume_fisier_specificatii* e fisierul de intrare
  - text sursa (implicit **stdin)**

e.g. (S.O. linux)

$ flex spec.lxi **=> lex.yy.c**

$ gcc lex.yy.c -o exe **=> exe**

$ ./exe **(<stdin)**

$ ./exe <fisier_intrare

Fisierul de specificatii

- fisier text
- Nume:

  poate avea orice nume acceptat de sistemul de operare. Din considerente istorice se prefera ca fisierul de specificatie sa aiba extensia **.lxi** sau **.l**
- Contine:
  - descrierea analizorului lexical
  - … cod C

Sectiunea de cod utilizator

- furnizam functia main
- specificam “pornirea analizorului lexical”,
  - apelam functia yylex
- de exemplu, in sectiunea cod utilizator:

```
…
%%
int main()
{
        yylex();
        …
}
```

Sectiunea de cod utilizator

Functia: **int yywrap()**

- spune ce facem dupa terminarea procesarii unui fisier (primeste indicatorul EOF)
  - yywrap <> 0 : dorim sa terminam procesarea
  - yywrap = 0 : dorim sa mai procesam …
- de exemplu, in sectiunea cod utilizator:

```
...
%%
int yywrap(){return 1;}
```

- Optiuni echivalenta cu: yywrap returneaza 1

```
%option noyywrap
```

Cateva variabile utile

char *yytext **-** reprezinta adresa zonei in care se depun caracterele atomului curent ;

int yyleng **-** reprezinta lungimea atomului curent;

FILE *yyin **-** desemneaza fisierul care contine textul sursa de analizat;

Mod de functionare:

- pentru sabloanele recunoscute se aplica actiunile corespunzatoare din specificatie (sectiunea de reguli),
  - genereaza text specific sau "*pot returna coduri lexicale*";
- textul din fisierul sursa care nu se potriveste cu nici un sablon este copiat automat in *yyout* (in cazul in care *yyout* nu a fost redefinit, el este implicit *stdout*);
- daca pentru un text se potrivesc mai multe sabloane, se alege potrivirea cea mai lunga;
- daca pentru un text se potrivesc mai multe sabloane de aceeasi lungime, se alege prima potrivire din specificatie, in ordine textuala;
- textul corespunzator potrivirii este copiat in variabila globala ***yytext*** iar lungimea acestuia este memorata in variabila globala ***yyleng***.
- Cand analizorul intalneste marcajul EOF, el verifica rezultatul functiei *yywrap()*. Rezultat fals (zero) inseamna ca analiza trebuie sa continue cu noul fisier desemnat de *yyin*. Un rezultat nenul al functiei *yywrap()* determina sfarsitul analizei si returnarea valorii 0 spre apelantul lui *yylex()*.