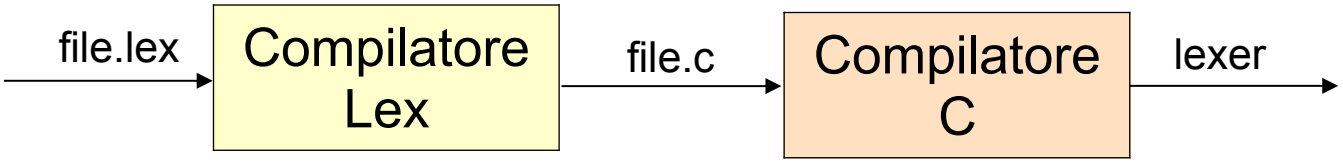
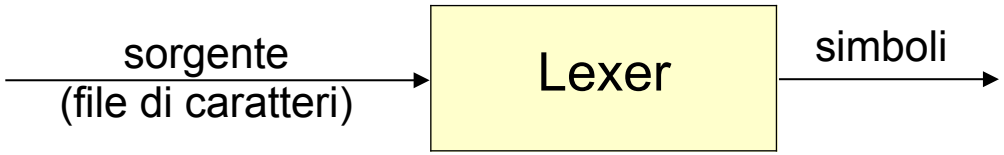
Lex (generatore di analizzatori lessicali)

- Ampia applicabilità: strumento che riconosce pattern di caratteri definiti da una expreg → poi ?

- Lex
  - linguaggio (expreg, azioni)
  - compilatore → codice C (*source-to-source*)



- Analizzatore lessicale:



Lex (ii)

- Specifica Lex (programma): 3 sezioni

Dichiarazioni
%%
Regole di traduzione
%%
Funzioni ausiliarie

- Dichiarazioni
  - black box (definizioni ausiliarie): **%{** #include, costanti, variabili **%}**
  - white box (definizioni regolari): *nome* *expr*

- Regole di traduzione

Expreg	*Frammento C*
r_1	{ $azione_1$ }
r_2	{ $azione_2$ }
r_n	{ $azione_n$ }

- Funzioni ausiliarie: necessarie alle azioni → compilabili anche separatamente

- Flex (GNU) → `yylex()` = funzione C in `file.c` che
  - riconosce il simbolo
  - esegue l'azione

Lex (iii)

- Notazione per la specifica di espressioni regolari:
  - Matching di stringhe di caratteri: `if` o `"if"`
  - Neutralizzazione dell'effetto del metacarattere: `"("` o `\(`
  - Uniformità con C per indicare caratteri di spaziatura 〈 `\t` `\n`
  - Meta-carattere • ≡ qualsiasi carattere ≠ `\n`
  - Interpretazione canonica dei metacaratteri `*` `+` `(` `)` `|` `?` (forma testuale)

    Esempio: " Stringhe di *a*, *b*, che iniziano con *aa* o *bb*, e terminano opzionalmente con *c* "

    Σ = {a,b,c} `(aa|bb)(a|b)*c?`
  - Matching dell'inizio di una linea (come primo carattere di una espressione regolare): **^**
  - Matching della fine di una linea (come ultimo carattere di una espressione regolare): **$**
  - Insieme complementare: **^** = primo carattere nel range `[^0-9abc]` = Σ – {0,1,...,9,a,b,c}

    Esempio: "Numeri in notazione scientifica" `("+"|"-")?[0-9]+("."[0-9]+)?(E("+"|"-")?[0-9]+)?`
  - Referenziazione di nomi di expreg: **{***nome***}**

```
nat     [0-9]+
snat    ("+"|"-")?{nat}
```

    ⋯▸ solo quando referenziato

Lex (ex1.lex)

Stampa delle linee precedute dal numero della loro posizione

```
%{
#include <stdio.h>
int pos = 1;
%}
%option  noyywrap
linea    .*\n
%%
{linea}  {printf("%d %s", pos++, yytext);}
%%
void main()
{
  yylex();
}
```

opzione di terminazione

definizione regolare

regola di traduzione

stringa lessicale (lexeme)

- Compilazione:

```
flex -o ex1.c ex1.lex
cc -o ex1 ex1.c
```

Lex (ex2.lex)

Stampa delle linee in posizione dispari (senza numeri)

```
%{
#include <stdio.h>
int pos = 1;
%}
%option  noyywrap
linea    .*\n
%%
{linea}  { if(pos++%2)
              printf("%s", yytext);
         }
%%
void main()
{
  yylex();
}
```

Lex (ex3.lex)

Sostituzione dei numeri dalla notazione decimale a quella esadecimale + stampa del numero di sostituzioni effettive

```
%{
#include  <stdio.h>
#include  <stdlib.h>
int cont  = 0;
%}
%option noyywrap
digit     [0-9]
num       {digit}+
%%
{num} { int n = atoi(yytext);
        printf("%x", n);
        if (n > 9) cont++; }
%%
void main()
{
   yylex();
   fprintf(stderr, "Tot sostituzioni = %d\n", cont);
}
```

Nota: Azione di default: quando un carattere (o una stringa di caratteri) non è parte di alcun simbolo

⇩

ECHO su output

Lex (ex4.lex)

Sostituzione dei numeri con valore >= 10 da notazione decimale a esadecimale + stampa del numero di sostituzioni

```
%{
#include  <stdio.h>
#include  <stdlib.h>
int cont  = 0;
%}
%option noyywrap
digit     [0-9]
num       {digit}{digit}+
%%
{num} { int n = atoi(yytext);
        printf("%x", n);
        cont++; }
%%
void main()
{
   yylex();
   fprintf(stderr, "Tot sostituzioni = %d\n", cont);
}
```

Generalizzazione per numeri che iniziano con zeri non significativi (00035):

```
digit    [0-9]
nonzero  [1-9]
num      0*{nonzero}{digit}+
```

Lex (ex5.lex)

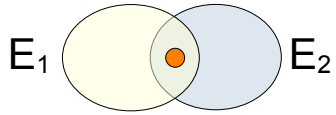
Stampa delle linee che iniziano o terminano con il carattere a

```
%{
#include <stdio.h>
%}
%option noyywrap
a_line    ^a.*\n
line_a    .*a\n
%%
{a_line} ECHO;
{line_a} ECHO;
.*\n      ;
%%
void main()
{
   yylex();
}
```

output di `yytext`

azione vuota

Note: Insieme ambiguo di regole (una stringa può corrispondere a diverse expreg, ad esempio: *a*)



Regole di priorità (built-in)

1. Principio del *maximal munch*.
2. Se ∃ diverse regole compatibili con la stringa → selezionata la regola specificata per prima.

```
.*\n          ;
{a_line}      ECHO;
{line_a}      ECHO;
```

output vuoto!

```
{a_line}      ECHO;
{line_a}      ECHO;
```

output = input!

Lex (ex6.lex)

Stampa delle linee che iniziano e terminano con il carattere a e contengono altri caratteri, tutti diversi da a

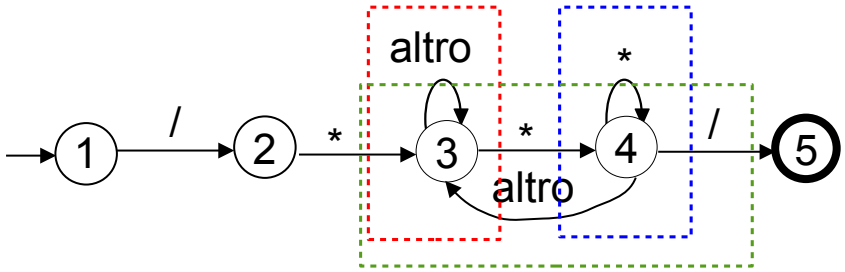
```
%{
#include <stdio.h>
%}
%option noyywrap
a_line_a    ^a[^a\n]+a\n
%%
{a_line_a} ECHO;
.*\n        ;
%%
main()
{
   yylex();
}
```

Lex (ex7.lex)

Sostituzione delle lettere maiuscole con minuscole, eccetto quelle nei commenti C

```
%{
#include <stdio.h>
#include <ctype.h>
#define FALSE 0
#define TRUE 1
%}
%option noyywrap
%%
[A-Z]     {putchar(tolower(yytext[0]));}
"/*"      {char c; int end = FALSE;
           ECHO;
           do { while ((c=input()) != '*')
                  putchar(c);
                putchar(c);
                while ((c=input()) == '*')
                  putchar(c);
                putchar(c);
                if(c == '/')
                  end = TRUE;
              } while(!end);
           }
%%
void main(){ yylex(); }
```

tecnica ibrida



`ba(b*(a*~(a|b)b*)*a*)*ab`

in cui: a = '*', b = '/'

Identificatori interni a Lex:

Identifier	*Description*
`yylex()`	Funzione di analisi lessicale
`yytext`	Stringa lessicale *(lexeme)*
`yyleng`	`strlen(yytext)`
`yyin`	File di input (default: `stdin`)
`yyout`	File di output (default: `stdout`)
`input()`	Input di un carattere ← `yyin`
`ECHO`	Azione di default: stampa `yytext` su `yyout`

Lex (ex8.lex)

Sostituzione lettere maiuscole con minuscole, eccetto quelle nei commenti Pascal

```
%{
#include <stdio.h>
#include <ctype.h>
%}
%option noyywrap
%%
[A-Z]       {putchar(tolower(yytext[0]));}
\{[^\}]*\}  ECHO;
%%
void main()
{
  yylex();
}
```

{ (~})*}

Lex (ex9.lex)

Conteggio di caratteri, parole e linee (wc); parola = lista di caratteri non bianchi

```
%{
#include <stdio.h>
int nc=0, nw=0, nl=0;
%}
%option noyywrap
word    [^ \t\n]+
eol     \n
%%
{word}   {nw++; nc+=yyleng;}
{eol}    {nl++; nc++;}
.        {nc++;}
%%
void main()
{
  yylex();
  printf("%d %d %d\n", nl, nw, nc);
}
```

spazio o tab

Lex: Analisi Lessicale

Riconoscimento dei simboli lessicali in un linguaggio di programmazione

```
%{
#include <stdlib.h>
#include "def.h"  /* IF, THEN, ELSE, ID, NUM, RELOP, LT, LE, EQ, NE, GT, GE */
int lexval;
%}
delimiter    [ \t\n]
spacing      {delimiter}+
letter       [A-Za-z]
digit        [0-9]
id           {letter}({letter}|{digit})*
num          {digit}+
%%
{spacing}    ;
if           {return(IF);}
then         {return(THEN);}
else         {return(ELSE);}
{id}         {lexval = store_id(); return(ID);}
{num}        {lexval = atoi(yytext); return(NUM);}
"<"          {lexval = LT; return(RELOP);}
"<="         {lexval = LE; return(RELOP);}
"="          {lexval = EQ; return(RELOP);}
"<>"         {lexval = NE; return(RELOP);}
">"          {lexval = GT; return(RELOP);}
">="         {lexval = GE; return(RELOP);}
%%
int store_id()  /* symbol table without keywords */
{ int line;
  if((line = lookup(yytext)) == 0) line = insert(yytext);
  return(line);
}
```