Rq : cette ...

Les expressions régulières Lex :

Une expression régulière de lex se compose de caractères normaux et de meta_caractères : . $ ^ [ ] ( ) { } < > + - * / \ ?

- si un meta_caractère doit faire partie explicitement de l'unité lexicale décrite par l'expression régulière elle doit être placée entre " ". ou être précédée du caractère par "\" ex «\+» représente le caractère + .

Le tableau suivant contient toutes les formes d'expressions régulières reconnues par Lex.

Rq c : représente un caractère qui ne fait pas partie des meta_caractères

r, r1, r2 : des expressions régulières

s : une chaine de caractères.

expression	reconnaît, accepte	exemple
c	tout caractère excepté un méta-caractère	a
"s"	la chaine de caractères s	"abc"
.	∀ caractère excepté le caractère "à la ligne"	a.b
^	comme premier caractère de l'expression régulière, le début de la ligne	^abcd
$	fin de ligne	abc$
[s]	n'importe lequel de caractère constituant la chaine s placée entre crochets	[abc]
[^s]	n'importe quel caractère à l'exception de ceux constituant la chaine s	[^abc]
r*	0 ou +rs occurrences de r	a*
r+	1 ou +rs occurrences de r	a+
r?	0 ou 1 occurrence de r	a?
r{m}	m occurrences de r	a{2}
r{m,n}	m à n occurrences de r	a{1,3}
r_1 r_2	r_1 suivie de r_2	ab
r_1|r_2	r_1 ou r_2	a|b
r_1/r_2	r_1 si elle est suivie de r_2	ab/cd
(r)	r	(a|b)?c
<x>r	r si lex se trouve ds l'état x	<x>abc

Les séquences d'échappement usuelles en C sont permises :

\n : à la ligne
\r : retour chariot
\t : "effacement arrière" (backspace) (tabulation)
\f : saut de page.
\nnn : denote le caractère dont la valeur ordinale est nnn en Octal.

[0-9] → l'ensemble des 10 chiffres.

ex [^\t\n]+ : toute chaine d'au moins un caractère ne comprenant pas de caractère blanc (espace, tabulation, fin de ligne)

exemple et une spécification ...

((([0-9]+)|([0-9]*.[0-9]+))([eE][-+]?[0-9]+)?

0 (nbre entier ou rien)

0 (sa partie entière)

5.900 e+00

3 types de nombre : 1 nbre entier

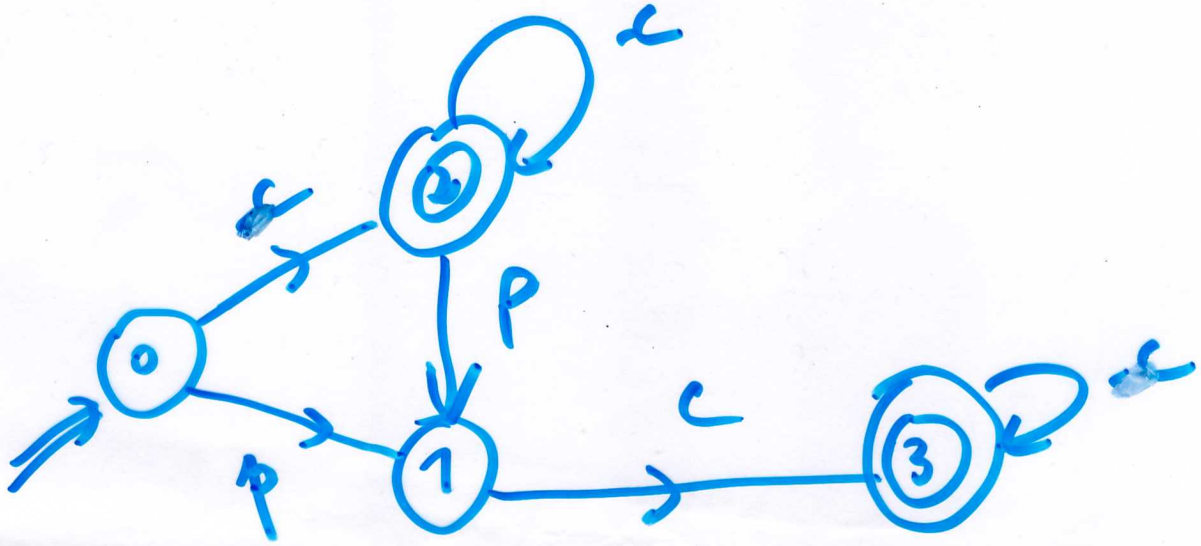
ou bien : 1 nbre décimal (sa partie entière ...)

ou bien : 1 nbre entier ou décimal suivi de l'exposant e

100 e-2

soit l'automate deterministe minimal acceptant l'expression reguliere

$p[c]^+ / [c]^+ / [c]^+ p[c]^+$



p c c

	c	p
→ e_0	e_2 x	e_1
e_1	e_3 x	pas defini
e_2	e_2	e_1
e_3	e_3 x	pas defini

```
PROGRAM automate_deterministe;

TYPE
   t_etat = (pas_defini, e0, e1, e2, e3);
   t_entree = (c, p);
   t_table_transition = array[t_etat, t_entree] of t_etat;
   t_etats_finaux = set of t_etat;

VAR
   table_transition : t_table_transition;
   etats_finaux : t_etats_finaux;
   etat_initial : t_etat;

Procedure initialisation;
BEGIN
   etats_finaux := [e2, e3];
   etat_initial := e0
   table_transition [e0, c] := e2; table transition [e0, p] := e1;
   table_transition [e1, c] := e3; table_transition [e1, p] := pas_defini;
   table_transition [e2, c] := e2; table_transition [e2, p] := e1;
   table_transition [e3, c] := e3; table_transition [e3, p] := pas_defini;
END;
FUNCTION accepte : boolean;
   var ch : char;
   etat_courant : t_etat;
      function lire_car : t_entree;
      begin
         read (ch);
         If ch = 'c' then lire_car := c
                     else lire_car := p
      end;
   Begin
      write ('Entrer la chaine à analyser : ');
      Etat_courant := etat_initial;
      while not eoln AND not (etat_courant = pas_defini) DO
         etat_courant := table transition [etat_courant, lire_car];
      accepte := etat_courant IN Etats_finaux
   end;

Begin
   initialisation;
   If accepte then writeln ('Phrase acceptée')
              else writeln ('phrase rejetée')
END.
```

Analyseur lexical de Leria par LEX

```
[a-zA-Z][a-zA-Z0-9_]*      Begin
                             echo;
                             If est_motclé(yytext, m_clé) then
                               return (m_clé)
                             else
                               return (identificateur)
                           END;

[0-9]+(“.”[0-9]*)?         Begin
                             echo;
                             return (C_numerique)
                           END;

'([^']|'')+                Begin
                             echo;
                             return (C_alphanumerique)
                           END;

“:=”                       Begin
                             echo;
                             return (affecte)
                           END;

“>=”                       Begin echo; return (superieur_egal) END;
“<=”                       Begin echo; return (inferieur_egal) END;
“<>”                       Begin echo; return (different) END;

[:,\.;=\>\[\]\(\)\<\-\+\*%\^]   Begin
                                  echo; returnc (yytext[1])
                                END;

“{”                        Begin
                             echo;
                             niveau:=1; Commentaire
                           END;

“\n”                       Begin
                             echo;
                             return (passer_a_la_ligne)
                           END;

[\t\f]
.  --->                    echo;
                           return (car_illégal);
```