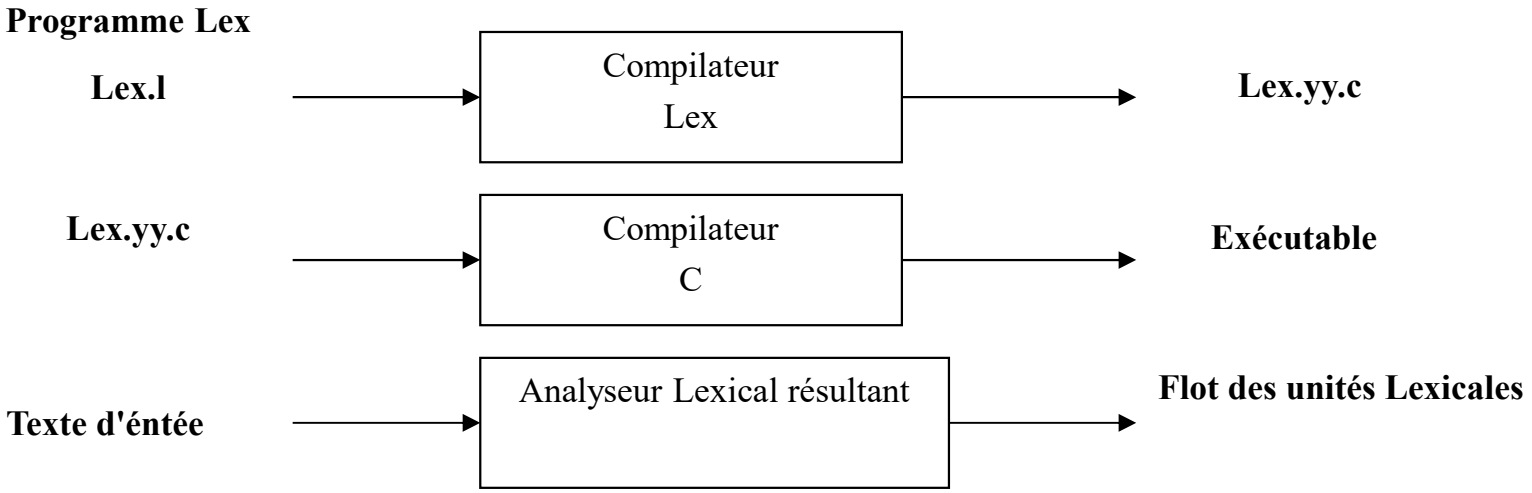
L'outil Lex:

Introduction

- Lex (fLex) est un utilitaire d'Unix.
- Lex accepte en entrée des spécifications d'unités Lexicales sous forme de **définitions régulières** et produit analyseur Lexical écrit en langage C.
- > Lex fichier.l

  donne le fichier Lex.yy.c qu'il faut compiler avec Lex :

  > gcc Lex.yy.c –l l
- Le fichier de spécifications Lex contient des expressions régulières suivies d'actions (règles de traduction).



Les expressions réguliers Lex

Expression	Signification	Exemple
c	tout caractère *c* qui n'est pas opérateur ou métacaractère	a
\c	caractère littéral c (lorsque c est un métacaractère)	\+ \.
"s "	chaîne de caractères	"bonjour "
.	n'importe quel caractère, sauf retour à la ligne (\n)	a.b
^	l'expression qui suit ce symbole débute une ligne	^abc
$	l'expression qui précède ce symbole termine une ligne	abc$
[s]	n'importe quel caractère de s	[abc]
[^s]	n'importe quel caractère qui n'est pas dans s	[^xyz]
r*	0 ou plusieurs occurrences de r	b*
r+	1 ou plusieurs occurrences de r	a+
r?	0 ou 1 occurrence de r	d?
r{m}	m occurrences de r	e{3}
r{m,n}	entre m et n occurrences de r	f{2,4}
r_1r_2	r_1 suivie de r_2	ab
$r_1\|r_2$	r_1 ou r_2	c|d
r_1/r_2	r1 si elle est suivie de r2	ab/cd
(r)	r	(a|b)?c
<x>r	r si Lex se trouve dans l'état x	<x>abc

Structure d'un fichier Lex

Un fichier de description pour Lex est formé de trois parties, selon le schéma suivant :

```
%{
    Déclaration en C des variables, des constants,...etc.
%}
    Déclaration des définitions régulières.
%%
    Expression régulières et actions correspondantes
%%
    Déclaration des procédures auxiliaires
```

Aucune partie n'est obligatoire.

Partie des déclarations

se composer de :

1. **Bloc littéral :**

- Commence par **%{** et se termine par **%}**. Où **%{** et **%}** doivent être placés en début de ligne.
- Contient des déclarations et définitions en C.
- Est copié tel quel dans le fichier Lex.yy.c produit par la commande Lex
- Les définitions et déclarations qu'il contient sont globales au programme produit

2. **Bloc des Définitions régulières:**

Ces définitions sont de la forme : ***notion*** ***expression régulière***

Exemple :

Separ [\t\n]
Lettre [A-Za-z]
chiffre [0-9]
ident {lettre}({lettre}|{chiffre})*
nbre {chiffre}+(\.{chiffre}+)?([Ee][\+\-]?{chiffre}+)?

Partie des productions

- Contient deux parties

1. **Partie gauche :**

- spécification des expressions régulières reconnues
- pour une chaîne de lettres et de chiffres, les guillemets peuvent être omis
- identificateurs d'expressions mis entre accolades

2. **Partie droite :**

- actions exécutées lorsque unités Lexicales reconnues
- actions définies avec syntaxe C
- si scanner appelé par parser YACC, alors :
    1- les attributs de l'unité Lexicale reconnue doivent être déposés dans **yylval**
    2- l'unité Lexicale reconnue doit être retournée

Partie des productions

- **Exemple**

```
[ \t\n]     { /* pas d 'action */ }
``<=``      {return(PPE);}
``<>``      {return(DIF);}
``<``       {return(PPQ);}
``=``       {return(EGA);}
``>=``      {return(PGE);}
``>``       {return(PGQ);}
``:=``      {return(AFF);}
 if         {return(si);}
 then       {return(alors);}
else        {return(sinon);}
{ident}     {strcpy(yylval,yytext); return(id);}
{nbre}      {strcpy(yylval,yytext); return(nb);}
```

Partie des productions

Remarques

En cas de conflit, Lex choisit toujours la règle qui produit le plus long lexème.

Exemple

prog action1

program action2

La deuxième règle sera choisie.

Si plusieurs règles donnent des lexèmes de mêmes longueurs, Lex choisit la première.

- *prog action1*
- *[a-z]+ action2* La première règle sera choisie.

Si aucune règle ne correspond au flot d'entrée, Lex choisit sa règle par défaut implicite :

- .|\n {ECHO} recopie le flot d'entrée sur le flot de sortie

Section Procédures auxiliaires

Section optionnelle, permet de :
- définir les fonctions utilisées dans les actions associées aux ERs reconnues
- définir (si nécessaire) le programme principal (main()).

Exemple: compter le nombre de voyelles, consonnes et de ponctuations d'un texte.

```
%{  int nbV = 0, nbC = 0, nbP = 0;
%}
consonne      [b-df-hj-np-tv-xz]
ponctuation   [,;:?!\.]
voyelle    [aeiouy]
%%
{voyelle}    nbV++;
{consonne}      nbC++;
{ponctuation}   nbP++;
.|\n           // ne rien faire
%%
main()
{ yylex();
printf("Il y a %d voyelles, %d consonnes et %d ponctuations.\n",nbV, nbC, nbP); }
```

Variables et fonctions prédéfinies

- **char yytext[ ]** : contient la chaîne de caractères qui a été acceptée.
- **int yyleng** : longueur de cette chaîne.
- **int yywrap()**: appelée en fin de flot d'entrée. Elle ne fait rien par défaut, mais peut être défini avec les fonctions auxiliaires. Elle retourne 0 si l'analyse doit se poursuivre sur un autre fichier d'entrée et 1 sinon.
- **int yylex()** : lance l'analyseur (et appelle yywrap()).
- **yylval:** retourne la valeur associé à l'unité lexicale reconnue**.**
- **ECHO :** affiche l'unité lexicale reconnue (⇔printf("%s",yytext))
- **FILE *yyout:** fichier de sortie. **FILE *yyin:** fichier d'entrée.
- **unput(char c)** : remet le caractère dans le flot d'entrée.
- **int yylineno** : numéro de la ligne courante.
- **yymore()**: concatène la chaîne actuelle avec celle qui a été reconnue avant
- **yyless(K>0)**: fonction admettant un entier comme argument, supprime les (yyleng-k) derniers caractères de yytext, c-à-d recule le pointeur de lecture sur le fichier d'entrée de **(yyleng-k)** positions.
- **yyterminate()** : fonction qui stoppe l'analyseur

Variables et fonctions prédéfinies

Exemple

Insert au début de chaque ligne son numéro.

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

Erreurs Lexicales

- Peu d'erreurs sont détectables au niveau lexical :
- Plusieurs stratégies sont possibles :

1. Mode panique : on ignore les caractères qui posent problème et on continue. Cette technique transfère le problème à l'analyseur syntaxique

2. Transformations du texte source : insérer un caractère, remplacer, échanger, etc. Elle se fait en calculant le nombre minimum de transformations à apporter au mot qui pose problème pour en obtenir un qui ne pose plus de problèmes.

➔ Cette technique de récupération d'erreur est très peu utilisée en pratique car elle est trop coûteuse à implanter.