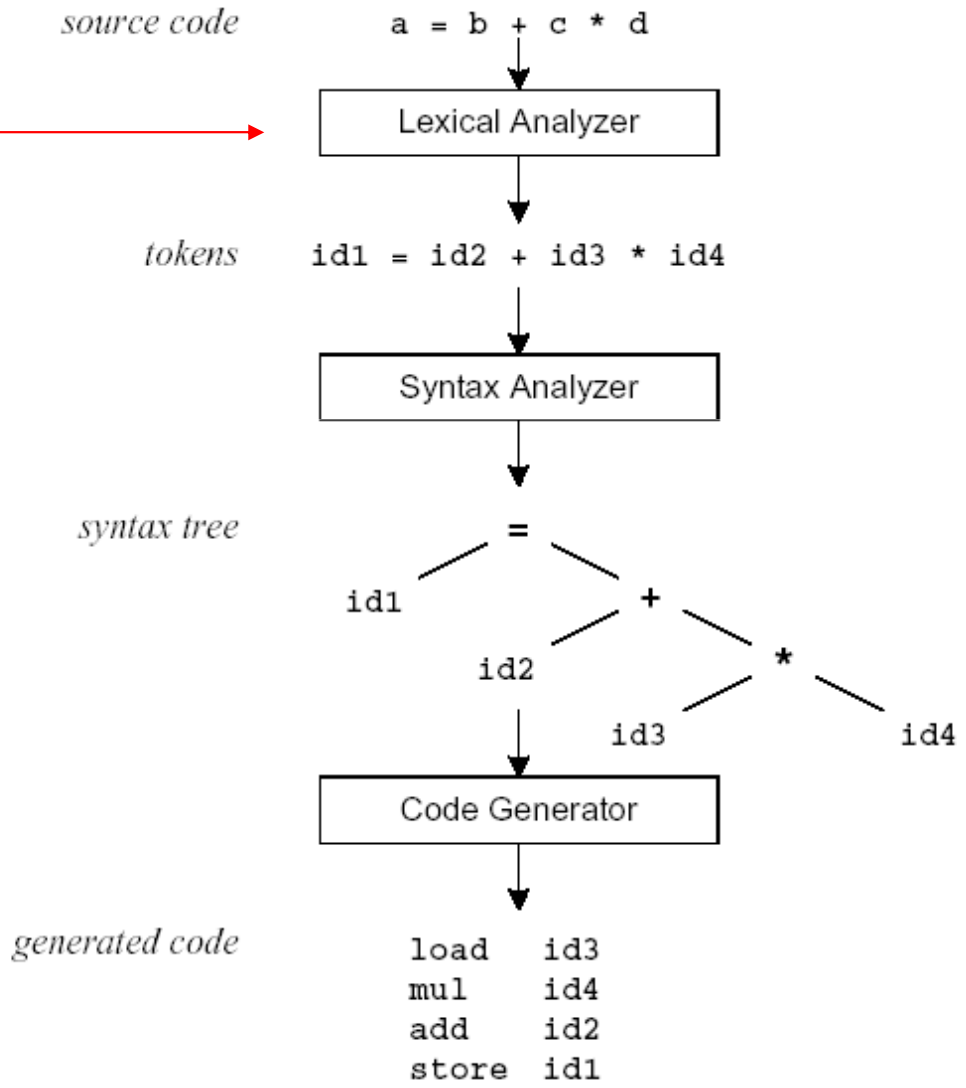
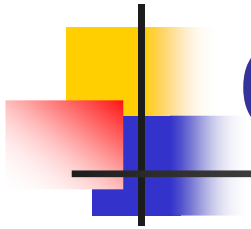
LEX
Générateur d'analyseurs lecicaux

ESI – École nationale Supérieure en Informatique

Octobre 2012

Séquence de Compilation





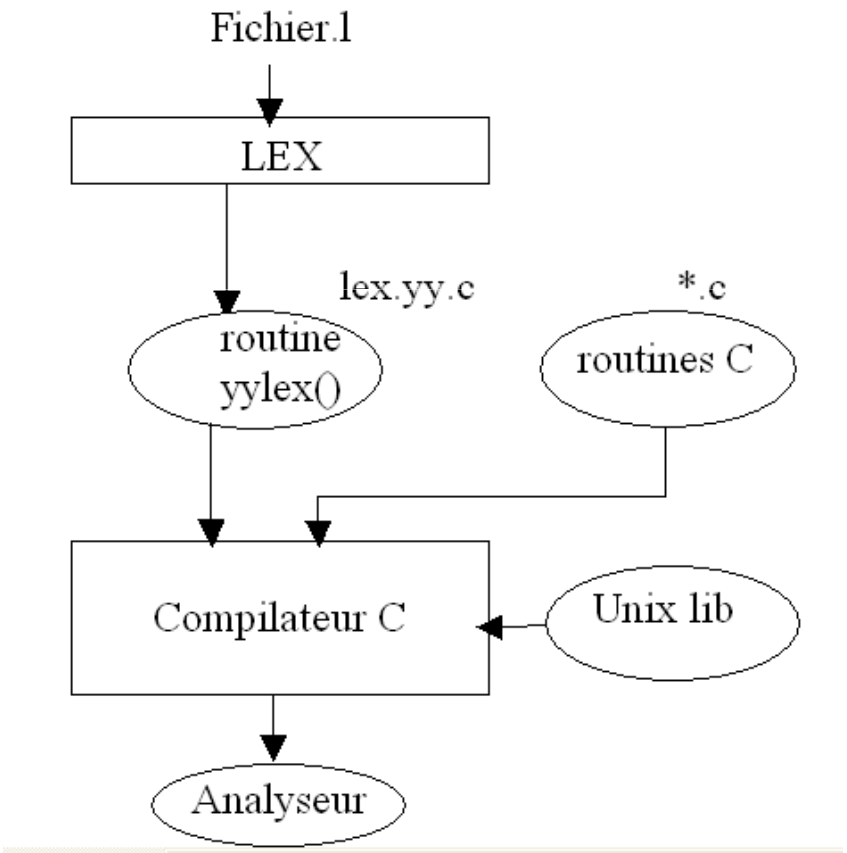
Qu'est-ce que Lex?

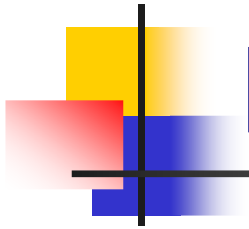
- LEX est un outil très populaire qui devient une norme de fait pour la génération d'analyseurs lexicaux.
- Lex :Aide à la génération rapide d'analyseurs.

Lex – Lexical Analyzer

- Prend en entrée les spécifications des utilisateurs sous forme d'expressions régulières et génère (s'il n y a pas d'erreurs) une programme en langage C.
- Ce programme en C contient la table de transition de l'automate d'états finis déterministe et l'algorithme d'analyse lexical.

Fonctionnement

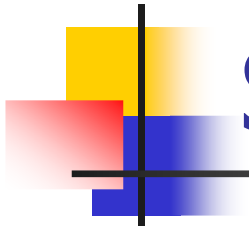




Programme Source Lex

- source Lex :
  - expressions régulières et
  - fragments de programme correspondants

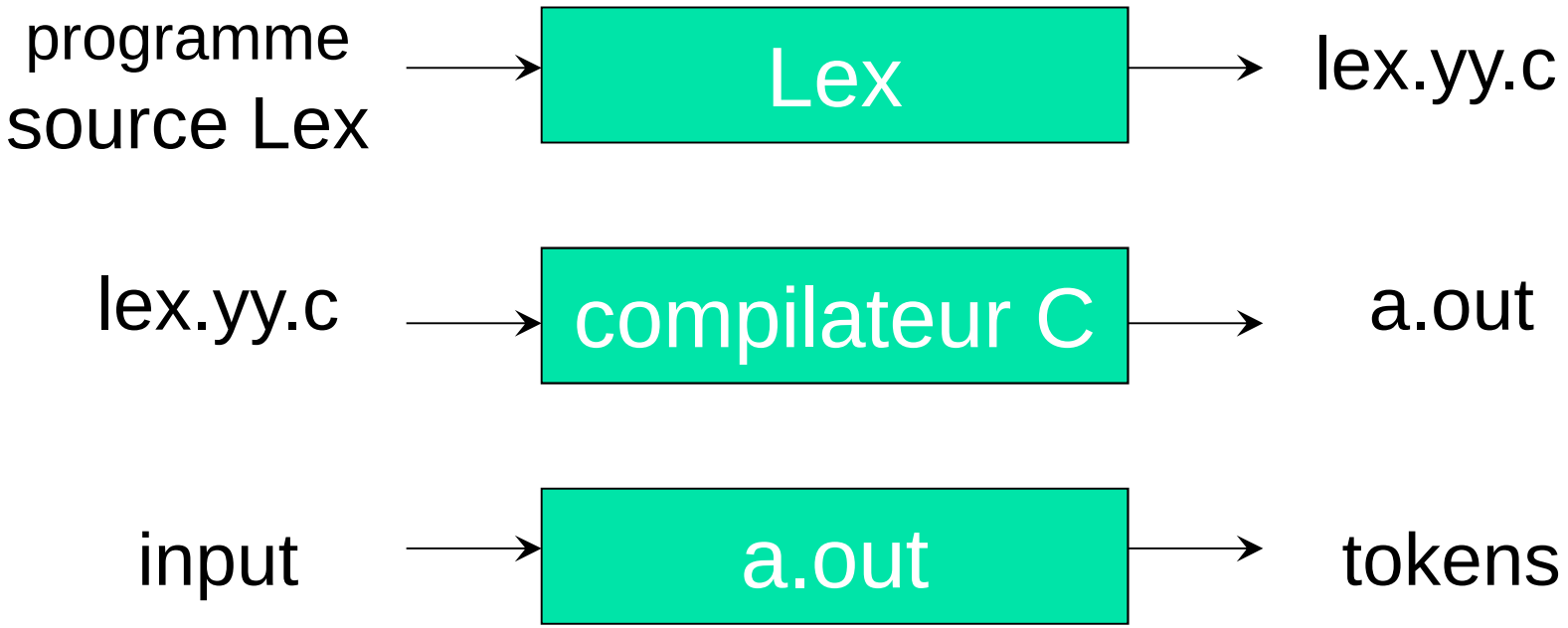
```
digit  [0-9]
letter [a-zA-Z]
%%
{letter}({letter}|{digit})*       printf("id: %s\n", yytext);
\n                                printf("new line\n");
%%
main() {
       yylex();
}
```



Source Lex au Programme C

- Programme C (lex.yy.c) qui :
  - Lit le flot d'entrée
  - Partitionne l'entrée en des mots correspondants aux E.R
  - copie sur un flot de sortie éventuel

Vue générale de Lex



Lex Source

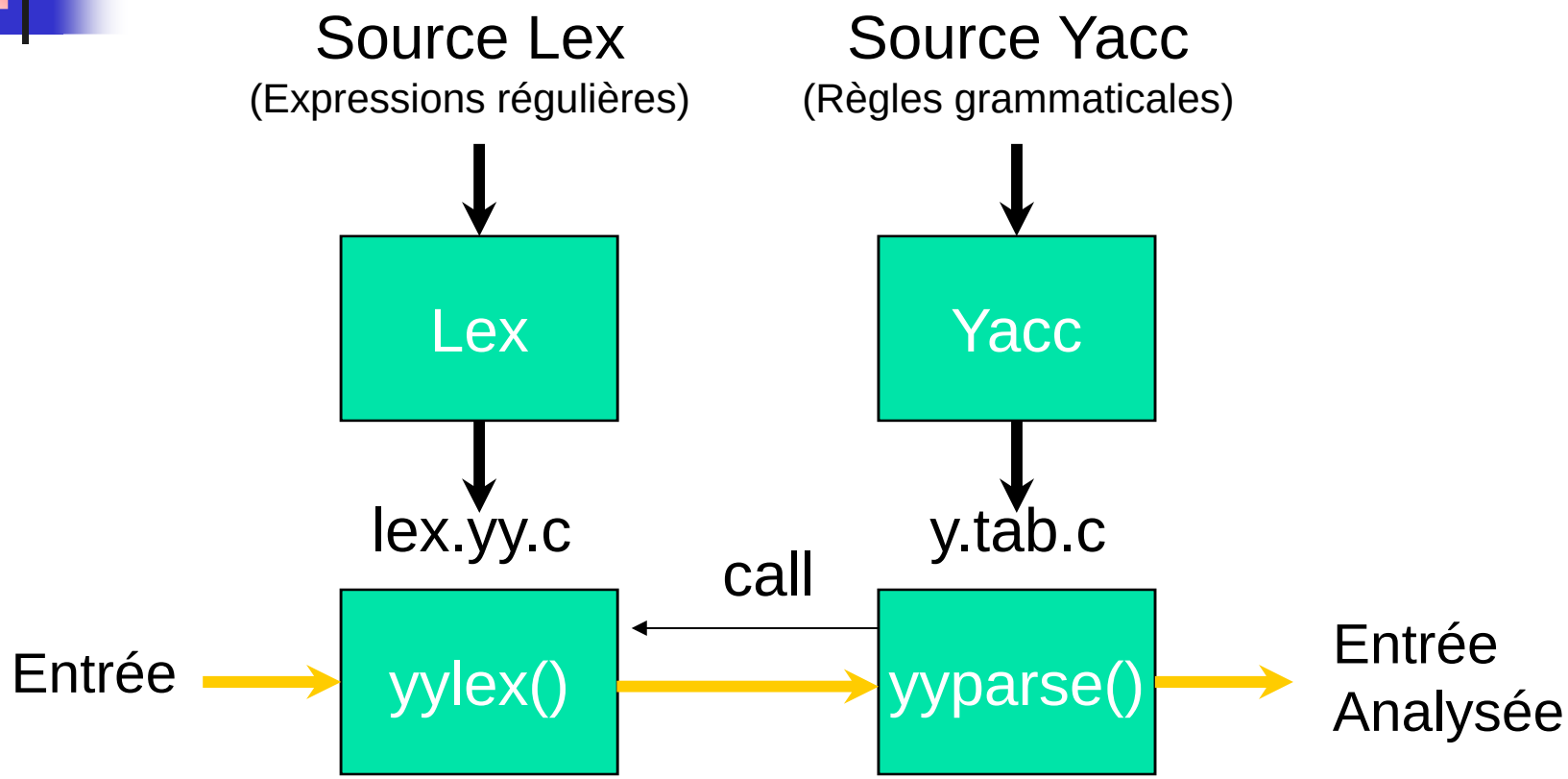
- trois sections séparés par les délimiteurs %%
- format géneral du source Lex :

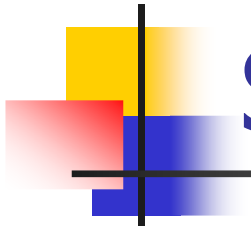
```
{déclarations}
%%                                  (obligatoire)
{règles de traduction}
%%                                  (optionnel)
{subroutines utilisateurs}
```

- Programme Lex minimum :

```
%%
```

Lex et Yacc

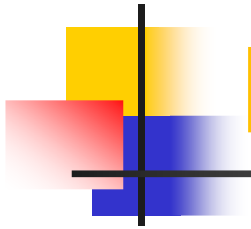

Source Lex
(Expressions régulières)
Source Yacc
(Règles grammaticales)
Lex
Yacc
lex.yy.c
y.tab.c
call
Entrée
yylex()
yyparse()
Entrée
Analysée



Section déclarations

- La section déclarations peut, elle-même, se composer de :
  - un bloc littéral
  - des définitions
  - des conditions de départ et états d'analyse

Section déclarations
Bloc littéral

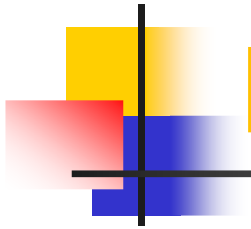


- commence par %{ et se termine par %}
- contient des déclarations et définitions en C
- est copié tel quel dans le fichier lex.yy.c produit par la commande lex
- les définitions et déclarations qu'il contient sont globales au programme produit par lex

Section déclarations
Bloc littéral

- %{
- #include "calc.h"
- #include <stdio.h>
- #include <stdlib.h>
- int k=0;
- %}

Section déclarations
Bloc définitions



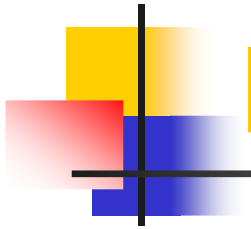
- Associations d'identificateurs à des expressions régulières
- Permettent de compacter l'écriture des expressions régulières

Section déclarations
Bloc définitions

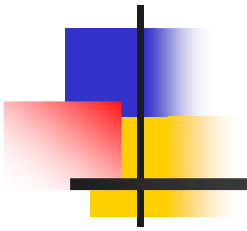
- separ [\t\n]
- espace {separ}+
- lettre [A-Za-z]
- chiffre [0-9]
- ident {lettre}({lettre}|{chiffre})*
- nbre {chiffre}+(\.{chiffre}+)?(E[+\-]?{chiffre}+)?

Utilisation des noms d'expressions régulières déjà définies entre accolades {}

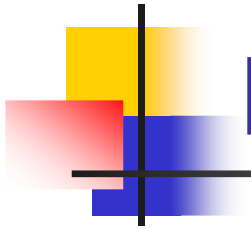
Section déclarations
Etats d’analyse



- Les états d’analyse permettent de définir plusieurs états de Lex.
- Lors de l'analyse d'un "chaîne" d'entrée Lex peut basculer d'un état à un autre en fonction du contexte.



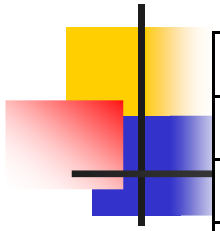
Expressions Régulières



Expressions Régulières Lex

- Une expression régulière correspond à plusieurs chaînes de caractèrs
- Expression régulière Lex
  - Alternative et groupage
  - Contexte
  - Répétitions and définitions
  - Operateurs
  - Classes de caractères
  - Caractère quelconque
  - expressions optionnels

Expressions Régulières LEX



Expression	Signification	Exemple
c	tout caractère c qui n'est pas spécial	a
\c	caractère littéral c lorsque c est un méta-caractère	\+ \.
"s"	chaîne de caractères	"bonjour"
.	n'importe quel caractère, sauf retour à la ligne	a.b
^	l'expression qui suit ce symbole débute une ligne	^abc
$	l'expression précédant ce symbole en fin de ligne	abc$
[s]	Un (01) caractère de la chaîne s	[abc]
[^s]	tout caractère qui n'est pas dans s sauf fin ligne	[^xyz]
r*	0, 1 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}
r1r2	r1 suivie de r2	ab
r1|r2	r1 ou r2	c|d
r1/r2	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

Opérateurs

“ \ [ ] ^ - ? . * + | ( ) $ / { }
% < >

- Si utilisés comme caractères :

  \$ = “$”

  \\ = “\”

Classes de caractères [ ]

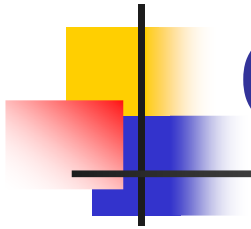
- `[abc]` reconnaît un (01) caractère a, b ou c
- Entre [ ], tous les caractères spéciaux sont ignorés sauf \ - et ^
- e.g.

  `[ab]` => a ou b

  `[a-z]` => a ou b ou c ou ... ou z

  `[-+0-9]` => tous les chiffres et signes + -

  `[^a-zA-Z]` => tout caractère qui n'est pas une lettre



Caractère quelconque .

- n’importe quel caractère, sauf retour à la ligne

Expressions Répétitions et option

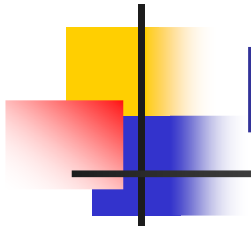
- a? => zero ou une instance de a
- a* => zero ou plusieurs instances de a
- a+ => une ou plusieurs instances of a

- E.g.

  `ab?c` => ac ou abc

  `[a-z]+` => tous les caractères minuscules

  `[a-zA-Z][a-zA-Z0-9]*` => Chaînes alphanumériques débutant par une lettre

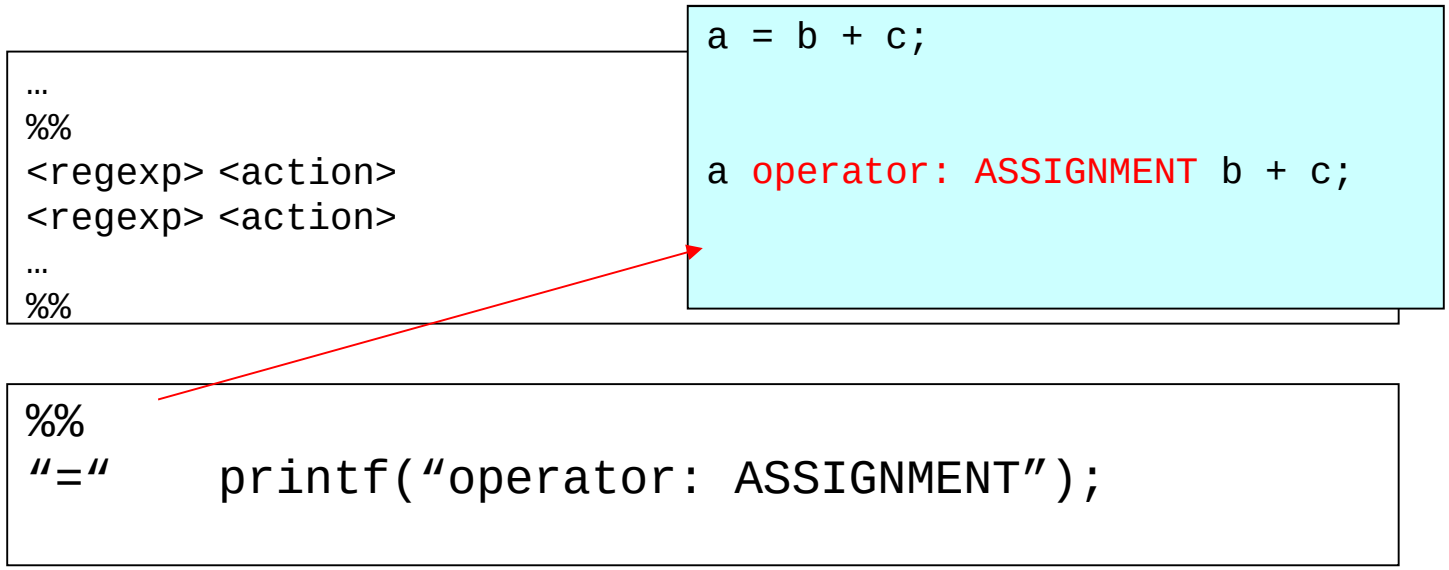


Priorités des opérateurs

- Précédence
  - Fermeture (*), ?, +
  - concaténation
  - alternative (|)
- Tous les opérateurs sont associatifs à gauche.
- Ex: `a*b|cd* = ((a*)b)|(c(d*))`

Section Règles de traduction

- Le source Lex est une table :
  - Expressions régulières et
  - fragments programme correspondants (actions)



Section Règles de traduction

- regexp <un ou plusieurs blancs> action (C code);
- regexp <un ou plusieurs blancs > { actions (C code) }

- Une instruction vide ; ignore l'entrée (pas d'action)

```
[ \t\n]          ;
```

  - Les trois séparateurs sont ignorés

```
a = b + c;
d = b * c;

↓↓

a=b+c;d=b*c;
```

Section Règles de traduction
Exemple 1

```
%%
[ \t]+$     ; /* Ne rien faire */
%%
main()
{
yylex() ;
}
```

Section Règles de traduction
Exemple 2

```
%{
int words, chars
%}
%%
[a-zA-Z]+                {words++ ;chars+=yyleng;}
%%
main()
{ yylex() ; }
```

Section Règles de traduction

- Quatre options spéciales pour les actions: |, ECHO;, BEGIN, et REJECT;
- | indique que l'action de cette règle est la même que la prochaine action
  - `[ \t\n]  ;`
  - `" "  |`
    `"\t"  |`
    `"\n"  ;`
- Les caractères qui ne correspondent à aucune expression régulières sont affichés ECHO en sortie

Section Règles de traduction

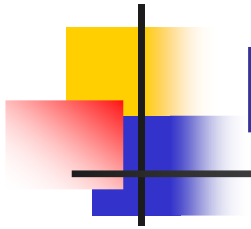
- REJECT
  - Aller faire la prochaine alternative

```
…
%%
pink        {npink++; REJECT;}
ink         {nink++; REJECT;}
pin         {npin++; REJECT;}
. |
\n          ;
%%
…
```

Variables Lex Prédéfinies

- yytext – une chaîne qui contient le lexème
- yyleng – La longueur du lexème
- yyin – pointeur du flux d'entrée
  - L'entrée par défaut de main() est**stdin**
- yyout -- pointeur du flux de sortie
  - La sortie par défaut main() est **stdout**.
- E.g.

```
[a-z]+          printf("%s", yytext);
[a-z]+          ECHO;
[a-zA-Z]+       {words++; chars += yyleng;}
```



Routines des librairies Lex

- yylex()
  - main() contient un appel à yylex()
- yymore()
  - fonction qui concatène la chaîne actuelle yytext avec celle qui a été reconnue avant
- yyless(n)
  - Garde les n premiers caractères dans yytext
- yywarp()
  - Est appelée quand Lex atteint la fin de fichier
  - Par défaut yywarp() retourne 1

Résumé Variables Lex Prédéfinies

Name	Function
`char *yytext`	pointer to matched string
`int yyleng`	length of matched string
`FILE *yyin`	input stream pointer
`FILE *yyout`	output stream pointer
`int yylex(void)`	call to invoke lexer, returns token
`char* yymore(void)`	return the next token
`int yyless(int n)`	retain the first n characters in yytext
`int yywrap(void)`	wrapup, return 1 if done, 0 if not done
`ECHO`	write matched string
`REJECT`	go to the next alternative rule
`INITAL`	initial start condition
`BEGIN`	condition switch start condition

Section Subroutines Utilisateur

- Programmation usuelle.

```
%{
        void foo();
%}
letter        [a-zA-Z]
%%
{letter}+    foo();
%%
…
void foo() {
        …
}
```

Section Subroutines Utilisateur

- Section main()

```
%{
        int counter = 0;
%}
letter [a-zA-Z]

%%
{letter}+     {printf(“a word\n”); counter++;}

%%
main() {
        yylex();
        printf(“There are total %d words\n”,
counter);
}
```

Rappel Utilisation sous Unix

- Fichier de spécification Lex scanner.l

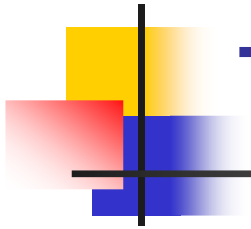
```
lex scanner.l
```

- Produit un fichier C appelé lex.yy.c qui contient l'analyseur.
- Compilation de lex.yy.c

```
cc lex.yy.c -ll
```

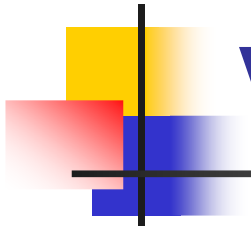
- Exécution

```
./a.out < inputfile
```



TP LEX à L'ESI

- Compilation de scanner.l sous Parser Generator
- Produit deux fichiers scanner.c et scanner.h
- Ouvrir un projet sous Visual C++ 2010 Express
- Compilation
- Execution



Versions de Lex

- Parser Generator
  http://www.bumblebeesoftware.com
- GNU -- flex
  www.**gnu**.org/software/**flex**/
- Jlex
  www.cs.princeton.edu/~appel/modern/java/**JLex**
  …

Références

- lex & yacc
  - by John R.Levine, Tony Mason & Doug Brown
  - O'Reilly
  - ISBN: 1-56592-000-7

- Mastering Regular Expressions
  - by Jeffrey E.F. Friedl
  - O'Reilly
  - ISBN: 1-56592-257-3