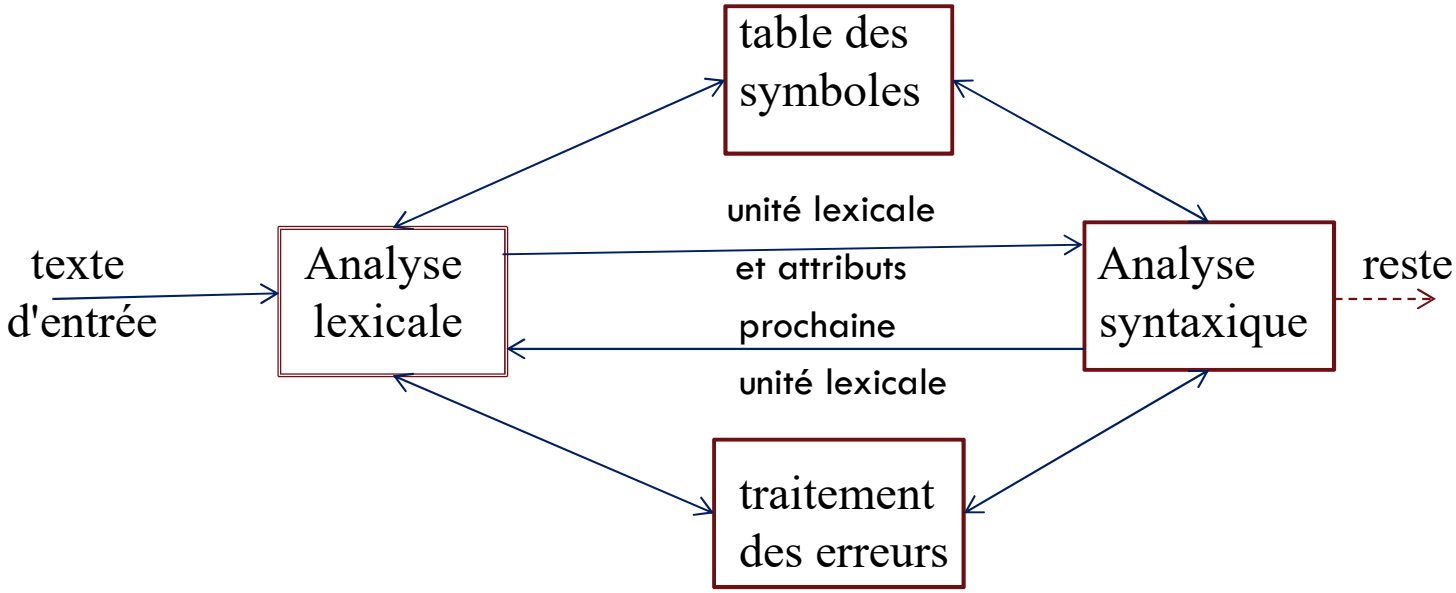
COMPILATION
EMSI - 4ÈME IIR
2017/2018

Prof. M. D. RAHMANI

L'analyse lexicale

1- Le rôle d'un analyseur lexical
2- Terminologie
3- Spécification des unités lexicales
    3.1- Chaînes et langages
    3.2- Opérations sur les langages
    3.3- Expressions régulières
    3.4- Définitions régulières
4- Reconnaissance des unités lexicales
5- Le langage FLEX
    5.1- Structure d'un programme FLEX
    5.2- Ecriture d'un analyseur lexical avec FLEX

1- Le rôle d'un analyseur lexical:



Interaction entre analyseur lexical et analyseur syntaxique.

2- Terminologie

- ✓ **`Unité lexicale:`** est un symbole terminal de la grammaire du langage.
- ✓ **`Modèle:`** est une règle qui décrit un ensemble de chaînes associées à la même unité lexicale.
- ✓ **`Lexème:`** est une suite de caractères du texte d'entrée qui concorde avec le modèle.

Exemple: **35** est un **lexème** (un mot) qui appartient à l'**unité lexicale** (la classe) **nombre**.

5- Le langage FLEX

Un programme écrit en langage ***LEX*** construit d'une manière automatique un analyseur lexical.

Il prend en entrée un ensemble d'*expressions régulières* et de *définitions régulières* et produit en sortie un *code cible en langage C*.

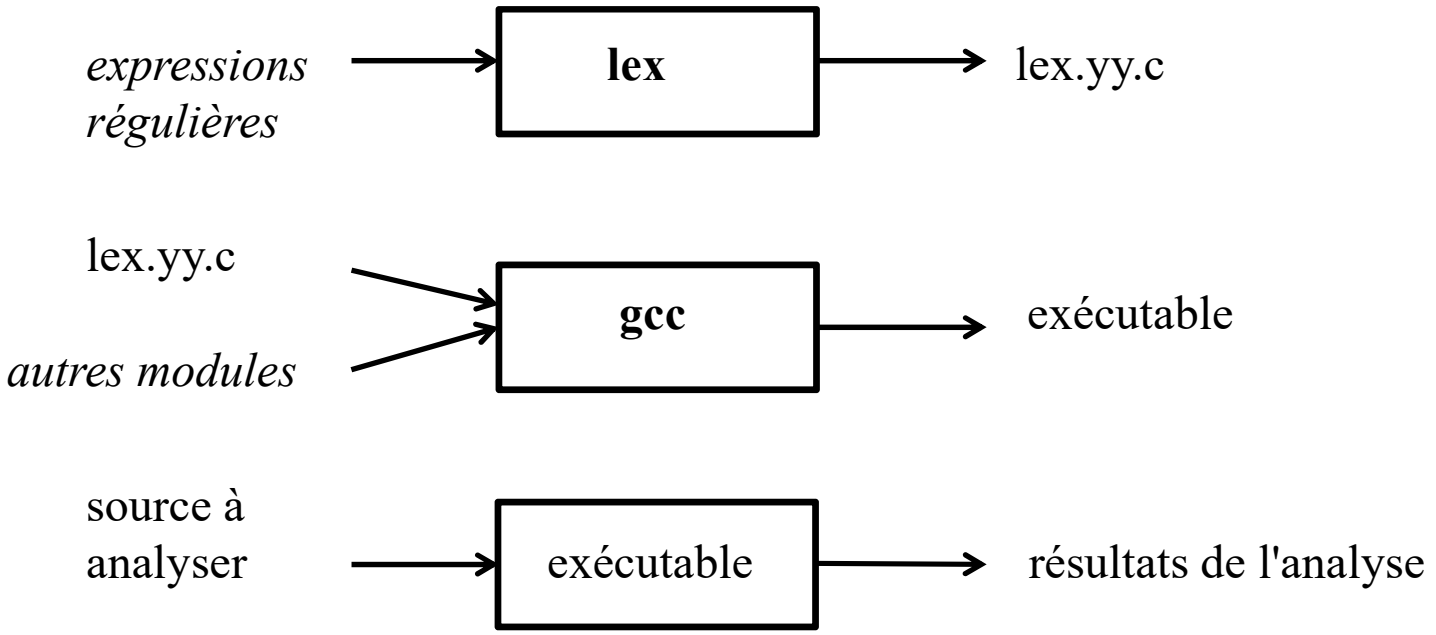
Ce code en langage C doit être compilé par le compilateur du langage C pour produire un exécutable qui est un analyseur lexical correspondant au langage défini par les expressions régulières d'entrée.

Plusieurs langages dérivés de LEX existent;

- Flex produit du C++
- JFlex produit du Java
- Lecl produit du Caml

5- Le langage FLEX

La compilation d'un code source en LEX se fait en deux étapes.



5- Le langage FLEX

5.1- Structure d'un programme FLEX:

Un programme source en langage lex est constitué de 3 sections délimitées par %%:

//1ère partie: déclarations

```
%{
   déclarations pour le compilateur C
%}
   définitions régulières
%%
```

// 2ème partie: règles de traduction

```
   expressions régulières + actions à réaliser
%%
```

// 3ème partie fonctions en C

```
   Fonctions annexes en langage C
```

5- Le langage FLEX

- La 1ère partie est constituée de:

  - déclarations en langage C des bibliothèques, variables, structures,...

  - déclarations de définitions régulières utilisables par les règles de traduction.

- La 2ème partie a la forme:

m_1	$\{action_1\}$	avec m_i une expression régulière ou une
m_2	$\{action_2\}$	définition régulière de la 1ère partie
...	....	et
m_i	$\{acion_i\}$	$action_i$ est l'action à réaliser par l'analyseur
...	....	lexical si un lexème est accepté par m_i.
m_n	$\{action_n\}$	

5- Le langage FLEX

- La 3ème partie est une suite de fonctions en C qui aident à l'analyse dans l'analyse par les règles de traduction de la 2ème partie.
  Cette partie peut contenir une fonction *main* du langage C.

Remarques:

- Le code doit commencer à la 1ère colonne.
- Le fichier LEX doit avoir l'extension ***.l*** ou ***.lex***

5- Le langage FLEX

5.2- Exemple d'un programme FLEX: espace.l

```
/* Un programme qui:
      - supprime une suite d'espaces et de tabulations d'un texte.
      - et bslama pour sortir
*/

%{
#include <stdio.h>
#include <stdlib.h>
%}
%%
bslama      {printf("\n\t Au revoir !\n"); exit(0);}
[\t ]+   {/* supprimer */
%%
int main() {
     printf("donner votre chaine :\n\t");
     yylex();
     return 0;
}

int yywrap() {  // le delimiteur de la lecture
     return 1;
}
```

5- Le langage FLEX

Remarques:

"**.**" est un opérateur qui veut dire tous caractère sauf le retour à la ligne.

"^" est un opérateur pour le complémentaire d'une classe.

`yytext` est un pointeur sur la chaîne analysée.

`yyin` correspond à l'entrée

`yylex()` est la fonction principale du programme écrit en LEX.

Options de compilation:

Avec LEX: 1/ *lex analex.l* produit *lex.yy.c*

2/ *cc lex.yy.c -ll* pour *library lex*

Avec FLEX: 1/ ***flex analex.l*** produit ***lex.yy.c***

2/ ***gcc lex.yy.c*** *(-lfl* pour *library fast lex facultatif)*

Expressions régulières du langage FLEX

Expression	Signification	Exemple
c	tout caractère 'c' qui n'est pas un opérateur	a
\c	caractère littéral 'c'	*
"s"	chaîne littérale s	"**"
.	tout caractère sauf fin de ligne	a.b
^r	r en début de ligne	^abc
r$	r en fin de ligne	abc$
[s]	tout caractère appartenant à s	[abc]
[^s]	tout caractère n'appartenant pas à s	[^abc]
[a-z]	tout caractère (lettre minuscule) entre 'a' et 'z'	d
[^a-z]	tout caractère qui n'est pas une lettre minuscule	5
r*	zéro ou plusieurs r	a*
r+	un ou plusieurs r	a+
r?	zéro ou un r	a?
r{m,n}	entre m et n occurrences de r	a{1,5}
r{3,}	trois r ou plus	b{3,}
r{2}	exactement deux r	c{2}
rs	r puis s	ab
r|s	r ou s	a|b
(r)	r	(a|b)
r/s	r quand suivi de s	abc / 123

5.3- Ecriture d'un analyseur lexical avec FLEX: *analex.l*

Définitions régulières des terminaux de la grammaire:

A noter qu'il faut reconnaitre les blancs aussi pour les ignorer.

```
delim   ⟶  espace|tabulation|fin_de_ligne
blanc   ⟶  (delim)+
IF      ⟶  si
THEN    ⟶  alors
ELSE    ⟶  sinon
operel  ⟶  <|<=|==|<>|>=|>
id      ⟶  [A-Za-z][A-Za-z0-9]*
nb      ⟶  (+|-)?[0-9]+(.[0-9])?((+|-)?(e|E)[0-9]+)?
```

Remarque: *Les commentaires et les blancs sont traités comme des modèles qui ne retournent aucune unité lexicale.*

5- Le langage FLEX

5.3- Ecriture d'un analyseur lexical avec FLEX: *analex.l*

1ère partie:

```
//déclarations en C
%{
#include<stdio.h>
%}
delim   [ \t\n]
bl      {delim}+
lettre [a-zA-Z]
chiffre [0-9]
id   {lettre}({lettre}|{chiffre})*
nb    (\+|\-)?{chiffre}+(\.{chiffre}+)?((e|E)(\+|\-)?{chiffre}+)?
% %
```

Remarque: Les caractères **+**, **-** et **.** sont précédés de **** pour les distinguer des opérateurs correspondants.

5- Le langage FLEX

2ème partie:

```
{bl}     {/* supprimer de la sortie */}
sinon     {printf("\n Mot clé: ELSE\n");}
si  {printf("\n Mot clé: IF\n");}
alors     {printf("\n Mot clé: THEN\n");}
{id}      {printf("\n Identificateur:%s\n",yytext);}
{nb}      {printf("\n Nombre:%s\n",yytext);}
"<="      {printf("\n Operateur relationnel: PPE\n");}
"<>"      {printf("\n Operateur relationnel: DIF\n");}
"<" {printf("\n Operateur relationnel: PPQ\n");}
">="      {printf("\n Operateur relationnel: PGE\n");}
"<" {printf("\n Operateur relationnel: PGQ\n");}
"=="      {printf("\n Operateur relationnel: EGA\n");}
"(" {printf("\n PO\n");}
")" {printf("\n PF\n");}
.   {printf("\n%s: Caractère non reconnu\n",yytext);}
%%
```

5- Le langage FLEX

3ème partie:

```
int main()
{
   FILE *fichier;
   printf("Nom du fichier à analyser");
   scanf("%s",&fichier);
   yyin=fichier;
   yylex();
   return 0;
}
```

Outils

Flex et bison: http://sourceforge.net/projects/winflexbison/

Nous aurons un fichier compressé: **`win_flex_bison-latest`** à décompresser.

JFlex: logiciel http://jflex.de/ , manuel: http://jflex.de/manual.html

et *Cup*: http://www2.cs.tum.edu/projects/cup/

Dev-C++: http://www.commentcamarche.net/download/telecharger-59-dev-c

Nous aurons un fichier exécutable: **`Dev-Cpp_5.9.2_TDM-GCC_4.8.1_Setup`**

Geany: http://www.geany.org/ , manuel: http://www.geany.org/Documentation/Manual