

## Manuel de FLEX

### 1. FLEX :

FLEX est un générateur d'Analyseur Lexical. Il est souvent utilisé avec un analyseur syntaxique. Un Analyseur Lexical est un programme permettant d'analyser un flux de chaînes de caractères et le segmenter en lexèmes (Tokens). Ces derniers représentent les entités lexicales d'un langage. A titre d'exemple : les identifiants de variables et de fonctions, les mots clés, les opérateurs, etc...

Un document FLEX comporte trois parties principales dont chacune d'elles est encadrée par le symbole « %% » :

#### 1.1. Première partie: Les expressions régulières

Cette section permet de donner les expressions régulières des différentes entités lexicales (lexème) de notre langage de programmation sous la forme suivante :

*<identificateur\_de\_l'entité><expression régulière>*

- *Un identificateur\_de\_l'entité* : Doit commencer par une lettre, et ne comportant que des caractères alphanumériques, des underscores ( \_ ) et des tirets ( - ).
- *Une expression régulière* : Doit être une *expression régulière* valide.

#### Exemple :

Entité lexicale                      Expression régulière

└───┬───┘ └───┬───┘

chiffre    [0-9]

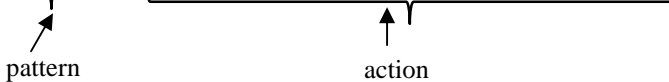
entier    {chiffre} + /\*utilisation d'une entité lexicale déjà définie\*/

## 1.2. Deuxième partie :Les règles de traduction

Cette section est capitale. Elle comporte l'ensemble des actions associées à chaque entité lexicale. Une règle de traduction est de la forme suivante :<pattern><action>

- **Un pattern**:c'est l'**expression régulière** décrivant un lexème.
- **Une action**:c'est le code C qui sera exécuté à chaque fois ou le lexème correspondant apparaît.

**Exemple :** {entier} {printf ("l'entité reconnu est un entier\n");}



## 1.2. Troisième partie : Code additionnel

C'est la dernière section d'un document FLEX. Elle contient le **post-code C**. Ce dernier représente le code que nous voulons exécuter. Il y sera recopié tel quel à la fin du fichier lex.yy.c.

**Exemple :**

```
// Déclarations C (pré-code)
% {
    # include <stdio.h>
% }

// Définitions

chiffre [0-9]
entier {chiffre}+
reel {chiffre}+"." {chiffre} *

%%

// Règles de traduction

{entier} {ECHO; printf ("\n");}
{reel} {printf ("%s", yytext); printf ("\n") ; }

%%

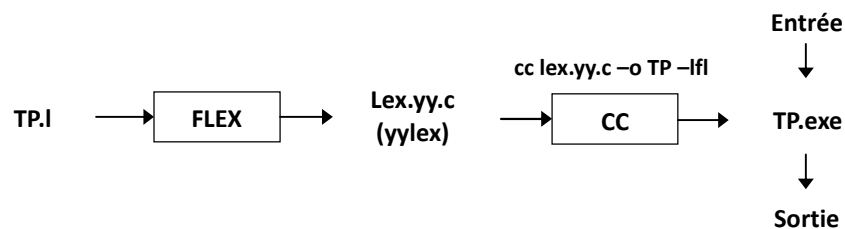
// Code Additionnel (post-code C)
intmain () {
    yylex ( ) ;
    return 0;
}
```

## 2. Macro-action de FLEX :

<i>Fonctions</i>	<b>yylex</b>	Permet de lancer l'analyseur lexical.
	<b>yywrap</b>	Elle est appelée par le <b>lexer</b> quand il rencontre la fin du fichier. Elle doit, soit obtenir un nouveau flux d'entrée et retourner simplement la valeur 0, soit renvoyer 1, signifiant que la totalité des flux a été consommée et que le lexer a fini sa tâche.
	<b>yyterminate</b>	Permet de provoquer la fin d'exécution du lexer.
	<b>ECHO</b>	Affiche l'unité lexicale reconnue (équivalente à <i>printf("%s", yytext)</i> )
<i>Variables</i>	<b>yytext</b>	Récupère le texte formant le lexème reconnu.
	<b>yylen</b>	Détermine la longueur du texte contenue dans <i>yytext</i>
	<b>yyval</b>	Est une variable globale utilisée par FLEX pour stocker la valeur correspondante au Token reconnu.
	<b>yylineno</b>	Est le numéro de la ligne courante.
	<b>yyin</b>	Fichier d'entrée.
	<b>yyout</b>	Fichier de sortie.

### 3. Commandes de Compilation :

FLEX lit un fichier de l'extension (\*.l), et génère un code C (lex.yy.c) pour la compilation.



**Pour compiler le programme :**    \$ flex TP.l  
   \$ cc lex.yy.c -o TP -lfl  
**Pour exécuter le programme :**    \$ ./TP  
**Pour arrêter le programme :**    \$ Ctrl+c

#### 4. Caractères spéciaux :

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

Pour utiliser ces caractères comme "caractères ordinaires", il faut les protéger en plaçant dans une chaîne entourée de double-quotes (") ou en les plaçant après un \. Les caractères \n, \t correspondent respectivement au saut de ligne et à la tabulation.

#### 5. Expressions régulières :

" : Une chaîne de caractères entourée par double-quotes représente la chaîne elle-même.

"**abc**" signifie la chaîne **abc**

[ ] : Une chaîne de caractères entre crochets représente un de ses éléments. Dans ce contexte, « | » et « - » indique un intervalle et ^ désigne l'exclusion.

[xyz] : x, y ou z

[a-zA-Z] : toutes les lettres minuscules et majuscules

[^0-9] : tous les caractères sauf les chiffres

. : Tout caractère sauf \n

| : Opérateur d'alternance

x|y|z : équivalent à [xyz]

[a-z][A-Z] : toutes les lettres minuscules et majuscules

\* et + : Opérateur de répétition (\* : zéro ou plusieurs fois, + : une ou plusieurs fois)

(x|y)\* : chaîne de longueur positive ou nulle constituée des caractères x ou y

[a-z]+ : chaîne de longueur strictement positive constituée de lettres minuscules

? : Opérateur d'occurrence zéro ou une fois

ab?c : chaîne **abc** ou chaîne **ac**

/ : Condition de reconnaissance

ab/cd : chaîne **ab** seulement si elle est suivie de la chaîne **cd**

**\$ et ^** : Début de ligne et fin de ligne

**ab\$** : chaîne **ab** en fin de ligne

**^ab** : chaîne **ab** seulement si elle est en début de ligne (après **\n** ou **\$**)

**{}** : Opérateur de répétition bornée - Définition

**a{1, 5}** : chaîne de longueur comprise entre 1 et 5 constituée du caractère **a**

**a{2, }** : chaîne de longueur supérieure ou égale à 2 constituée du caractère **a**

**a{2, 2}** : chaîne de longueur 2 constituée du caractère **a**

**a{digit}** : chaîne prédéfinie de nom **digit**

Expression	Exemple
<b>abc</b>	abc
<b>abc*</b>	ab, abc, abcc, abccc, ...
<b>abc+</b>	abc, abcc, abccc, ...
<b>a(bc)+</b>	abc, abcbc, abcbcbc, ...
<b>a(bc) ?</b>	a, abc
<b>[abc]</b>	a, b, c
<b>[a-z]</b>	a, b, c, d, ... z
<b>[a\~z]</b>	a, -, z
<b>[-az]</b>	-, a, z
<b>[a-zA-Z0-9]+</b>	Un ou plusieurs caractères alphanumériques
<b>[ \t\n]+</b>	Espaces
<b>[^ab]</b>	Tous les caractères sauf : a, b
<b>[a^b]</b>	a, ^, b
<b>[a b]</b>	a,  , b
<b>a b</b>	a ou b