TP1 : Flex (Introduction)

1. Rappel :

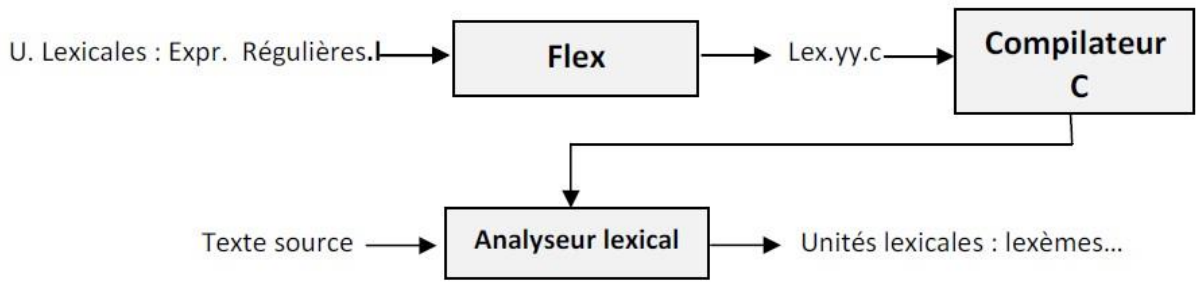
La tâche principale d'un analyseur lexical est de lire un texte source (suite de caractères) et de produire comme résultat une suite d'unités lexicales. La reconnaissance des unités lexicales est basée sur la notion d'expressions régulières. Théoriquement, la construction d'un analyseur lexical consiste à :

- Définir les unités lexicales.
- Modéliser chaque unité lexicale par une expression régulière.
- Représenter chaque expression régulière par un diagramme de transition (automate).
- Construire le diagramme global.
- Implémenter à la main le diagramme obtenu.

Généralement, l'implémentation à la main d'un diagramme avec un grand nombre d'états est une tâche qui n'est pas assez facile. En outre, si on a besoin d'ajouter ou modifier une unité lexicale, il faut parcourir tout le programme pour effectuer les modifications nécessaires. Plusieurs outils ont été bâtis pour simplifier cette tâche. (Flex par exemple)

1. Présentation de l'outil Flex :

L'outil Flex (version GNU de LEX) est un générateur d'analyseurs lexicaux. Il accepte en entrée des unités lexicales sous formes d'expressions régulières et produit un programme écrit en langage C qui, une fois compilé, reconnaît ces unités lexicales. L'exécutable obtenu lit le texte d'entrée caractère par caractère jusqu'à l'identification de plus long préfixe du texte source qui concorde avec l'une des expressions régulières.



Un fichier de spécifications Flex se compose de quatre parties :

%{
Les déclarations en c
%}

Déclaration des définitions régulières

%%
Règles de traduction
%%

Bloc principal et fonctions auxiliaires en C

Définitions régulières :
Une définition régulière permet d'associer un nom à une expression régulière et de se référer par la suite dans la section de règles à ce nom, plutôt qu'à l'expression régulière.

Règles de traduction :

exp_1 **{** *action*$_1$**}** exp_2
{ *action*$_2$**}**
… **…** exp_n **{**
action$_n$**}**

Chaque ***expi*** est une expression régulière qui modélise une unité lexicale. Chaque ***actioni*** est une suite d'instructions en C.

Variables **:** ***yyin****: fichier de lecture (par défaut: stdin)* ***yyout*** *: fichier d'écriture (par défaut: stdout)* ***char yytext []*** *: tableau de caractères qui contient le lexème accepté.* ***int yyleng*** *: la longueur du lexème accepté.*

Fonctions **:**
int yylex ( ) *: fonction qui lance l'analyseur.*
Int yywrap ( ) *: fonction qui est toujours appelée à la fin du texte d'entrée. Elle retourne 0 si l'analyse doit se poursuivre et 1 sinon.*

Expressions régulières Flex :

c	*le caractère c*
.	*tout caractère seul, sauf retour à la ligne*
[ ]	*un des caractères spécifiés [abcdABCD0123]*
-	*un intervalle de caractère si dans [ ] [a-dA-D0-3]*
?	*0 ou 1 occurrence 10?9 est 109 ou 19*
*******	*répétition (zéro ou plus) [ab]**
+	*répétition (au moins une) [ab]+ est [ab][ab]**
|	*alternative 000|110|101|011*
( )	*groupement d'expressions (0|2|4|8)**

{ } *permet de définir des répétitions if (def)? est if(def)\{0,1}*

$ *comme le dernier caractère de l'expression, il signifie la fin de la ligne.*

^ *comme le premier caractère de l'expression, il signifie le début de la ligne ou le complément dans [ ]* \"[^\"\n]*\"

2. Premier pas avec Flex :

1. Commencer par l'installation des outils *Flex* et *CodeBloks* (outils de développement utilisant le langage C).
2. Ecrire le programme Flex suivant permettant de dire si une chaîne en entrée est un nombre binaire ou non : Ouvrir un nouveau fichier texte et taper le code ci-dessus. Le fichier doit être enregistré avec l'extension **.l** (exemple : *binaire.l*)

```
%%
(0|1)+ printf ("c'est un nombre binaire");
.* printf ("ce n'est pas un nombre binaire");
%%
int yywrap(){return 1;}
main() { yylex();
}
```

3. Placer le fichier obtenu dans '*C:\Program Files\GnuWin32\bin* '
4. A partir de l'invite de commande, lancer la commande : *C:\Program Files\GnuWin32\bin>* ***flex binaire.l***
5. En cas de réussite, le fichier *lex.yy.c* est généré dans le même répertoire.
6. Compiler le fichier *lex.yy.c* avec *CodeBloks* pour générer l'exécutable.
7. Tester le fichier *lex.yy.exe* obtenu pour vérifier que ça fonctionne correctement.

3. Exercices :

1. Modifier l'exercice précédent pour qu'il n'affiche que les nombres binaires reconnus.
2. Ecrire et compiler le fichier de spécifications suivant :

```
 pairpair (aa|bb)*((ab|ba) (aa|bb)*(ab|ba)
(aa|bb)*)*
%%
{pairpair} printf ("[%s]: nombre pair de a et de b\n", yytext);
a*b* printf ("[%s]: des a d'abord et des b ensuite\n", yytext);
. %%
int yywrap() {return 1;}
main() { yylex();
}
```

3. Tester les entrées babbaaab abbb aabb baabbbb bbaabbba baabbbbab aaabbbba.
4. Même question en permutant les deux lignes :

```
 a*b* printf("[%s]: des a d'abord et des b ensuite \n",yytext);
(pairpair) printf ("[%s]: nombre pair de a et de b \n",yytext);
```

5. Y'a t-il une différence ? Laquelle ? Pourquoi ?
6. On considère l'unité lexicale **id** définie comme suit : un identificateur est une séquence de lettres et de chiffres. Le premier caractère doit être une lettre. Ecrire à l'aide de Flex un analyseur lexical qui permet de reconnaître à partir d'une chaîne d'entrée l'unité lexicale **id.**

7. Modifier l'exercice précédent pour que l'analyseur lexical reconnaisse les deux unités lexicales **id** et **nb** sachant que **nb** est une unité lexicale qui désigne les entiers naturels.