Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique
Département d'Informatique

License 3 : ISIL Module Compilation

Manuel FLEX

1. Flex

FLEX est un générateur d'analyseur lexical. Ce dernier est un programme permettant de reconnaitre les mots d'un langage donné. Etant donné un flux de chaines de caractères écrit dans un langage donné, l'analyseur lexical permet de segmenter le flux en lexèmes représentant les entités lexicales de ce langage comme les identifiants de variables, les mots clés, les opérateurs, etc.

Afin de générer l'analyseur lexical d'un langage « L », Flex prend en entré un fichier descriptif du langage (ayant l'extension « .l ») et génère un fichier (nommé par défaut « lex.yy.c ») qui contient le code source (en C) de l'analyseur lexical. Il suffit de compiler ce code source pour avoir l'exécutable de l'analyseur lexical (voir figure 1).

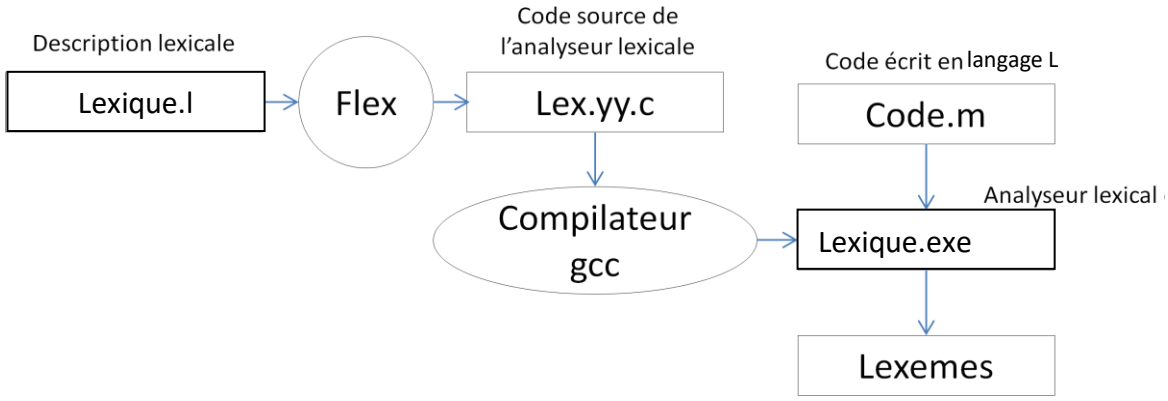

Figure1 : étapes de génération de l'analyseur lexical en utilisant Flex

2. Structure du fichier de description lexicale (*.l)

Le fichier d'entré est découpée en 4 zones : séparées par les balises %{, %}, %%, %%. (Voir figure 2)

```
%{
 // Déclarations C (pré-code)
%}
// Déclaration des définitions régulières
%%
// Règles de traduction
%%
// Code Additionnel (post-code C)
```

Figure 2 : Structure du fichier de description lexicale

2.1. Déclarations C (pré-code)

Cette partie contient toutes les déclarations (en langage C) des bibliothèques, variables, structures, unions, etc. Exemples : #include <stdio.h> ; #define ENDOFUTF 8, etc.

2.2. Déclarations des définitions régulières

Dans cette partie, on définit les entités lexicales du langage (lexèmes) ayant une forme régulière en utilisant ce qu'on appelle une expression régulière ou rationnelle. Ces définitions sont données sous la forme suivante :

<idf_entité> <espace(s)> <exp_régulière>

<idf_entité> : est le nom qu'on donne à l'entité lexicale.
<exp_regulière> : l'expression régulière qui génère les formes possibles de cette entité.
Exemple : les nombres entiers positifs sont des entités lexicales dont la forme est une suite infinie de chiffres (de 0 à 9). Exemple : 1, 10, 125, 196, 1000, 102409, etc. Ceci peut être traduit par l'expression régulière suivante :

<idf_entité> <espace(s)> <exp_regulière>
Entier_positif **[1-9][0 -9]***

Une entité définie peut être utilisée dans d'autres définitions régulières et/ou dans la partie des règles de traduction. Elle doit être mise entre accolades dans les deux cas.
Exemple:
chiffre [0−9]
Entier_positif [1-9]{chiffre}*

2.3. Règles de traduction

Cette section contient l'ensemble des actions associées à chaque entité lexicale.
Une règle de traduction est de la forme suivante :<entité><espace(s)> <action>
<entité> : elle peut correspondre à :

- une expression régulière d'une entité lexicale : **[0 -9] +**
- un nom d'entité lexical déjà défini dans la partie de définitions : {chiffre} (les accolades sont obligatoires)

<action> : c'est du code C mis entre accolade. Cette partie sera exécuté à chaque fois que le lexème correspondant trouvé.
Exemples :

[0 -9] {printf ("Je viens de lire un chiffre\n"); /*autres instructions*/}
{Entier_positif} {printf ("Je viens de lire un entier positif\n"); /*autres instructions*/}

2.4. Code additionnel (post-code C)

Cette dernière partie du fichier contient le ***post-code C***. C'est une suite de fonctions en C qui aident à l'analyse par les règles de traduction de la 2ème partie. Cette partie peut contenir une fonction main du langage C.

3. Les expressions régulières

Les expressions régulières sont basées sur caractères spécifiques. Le tableau suivant donne une liste (non-exhaustive) de ces caractères.

"	Une chaîne de caractères entourée par guillemets représente la chaîne elle-même.	**"abc"** signifie la chaîne **abc**
[ ]	Une chaîne de caractères entre crochets représente un de ses éléments. Dans ce contexte, «-» 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

|	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)	**ab* = a,ab,abb,…** **ab+ = ab,abb,…** **[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	**a{1, 5} :** chaîne constituée de une à 5 lettres « a » **a{2, } :** chaîne constitué de deux lettres « a » ou plus.
( )	Utiliser pour limiter un opérateur sur une partie précise	**a(bc)?** a, abc
.	Tous les caractères	

Déspécialisation des caractères spéciaux

Afin d'utiliser les caractères spéciaux du Flex (" \ [ ] ^ - ? . * + | ( ) $ / { }) en tant que caractères ordinaires, il faut les protéger en plaçant dans une chaîne entourée de double-quotes (") ou en les plaçant après un \ .

Note : Les caractères \n, \t correspondent respectivement au saut de ligne et à la tabulation.

4. **Quelques éléments de FLEX :**
   - **Variables :**
     - **yytext :** variable contenant le lexème reconnu.
     - **yyleng :** détermine la longueur du texte contenue dans *yytext*
     - **yylval :** variable globale utilisée par FLEX pour stocker la valeur correspondante au lexème reconnu.
     - **yylineno :** le numéro de la ligne courante.
     - **yyin :** fichier d'entrée.
     - **yyout :** fichier de sortie.
   - **Fonctions et macros :**
     - **yylex : p**ermet de lancer l'analyseur lexical.
     - **yywrap : e**lle est appelée par l'analyseur lexical 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 l'analyseur a fini sa tâche.
     - **yyterminate : p**ermet de provoquer la fin d'exécution de l'analyseur.
     - **ECHO** Affiche l'unité lexicale reconnue (équivalente à *printf ("%s", yytext)*)

5. Premier pas avec Flex

Supposons que nous voulons générer un analyseur lexical qui reconnait les nombres entiers naturels et déclenche une erreur lorsque le mot lu n'est pas reconnu.

5.1.Fichier « Entier_nat.l »

```
%{
#include <stdio.h>
%}
chiffre       [0-9]
Entier_nat       0 | [1-9]{chiffre}*

%%
{Entier_nat}  {printf("le mot %s est reconnu", yytext);}
.             {printf("\nErreur lexical: le mot %s nest pas reconnu",yytext);}
%%
int main()
{
   yylex();
   return 0;
}
int yywrap(void)
{
    return 0;
}
```

5.2.Génération de l'analyseur lexical

Il faut suivre la procédure suivante pour générer l'analyseur lexical

Commande	Explication	Sortie
$ flex Entier_nat.l	Exécuter flex avec le fichier « Entier_nat.l » en entré	Un fichier lex.yy.c qui contient le code source de l'analyseur lexical.
$ cc lex.yy.c –o Entier_nat	Compiler le code source généré en utilisant un compilateur C (cc ou gcc). Donner un nom à l'exécutable ave l'option –o nom	Le fichier exécutable de l'analyseur lexical nommé Entier_nat
$ ./Entier_nat	exécuter le programme et entrer des entiers naturels	Affichage des entités lexicales reconnues ou déclencher une erreur
$ Ctrl+c	Arrêter le programme	

5.3.Exemples d'exécution de l'analyseur lexical

Mot introduit	Affichage
1	Le mot 1 est reconnu
99	Le mot 99 est reconnu
a	Erreur lexicale : le mot a n'est pas reconnu
!	Erreur lexicale : le mot ! n'est pas reconnu