Introduction

Fonctionnement de FLEX

Le générateur FLEX est un générateur d'analyseurs lexicaux. il génère un super-automate à partir d'un ensemble d'expressions régulières.

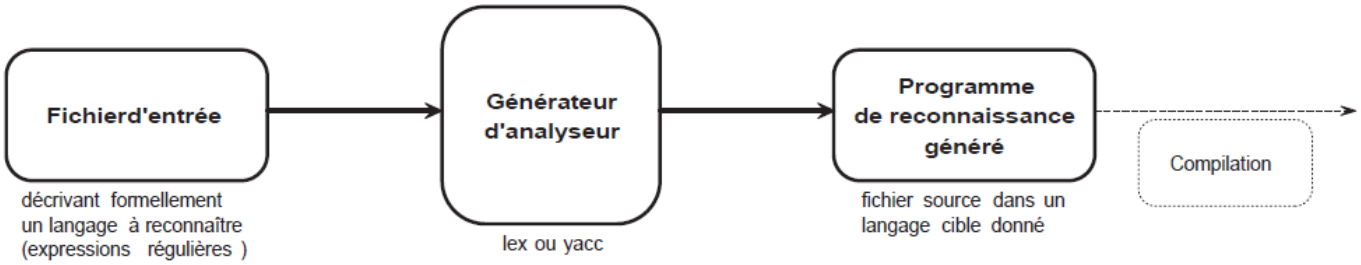

FIGURE 1 – Principe d'utilisation des outils Lex & Yacc.

Le programme produit est un autre programme source écrit en langage C « `lex.yy.c` ».
Il contient le sous-programme `yylex()`. Pour avoir l'exécutable, utiliser le compilateur C (GCC), en faisant l'édition de liens avec la bibliothèque de FLEX (le nom de la bibliothèque est **fl**).

```
flex fichierSource.lex
gcc -o fichierExecutable lex.yy.c -lfl
```

Les expressions régulières

Une ER FLEX est constituée d'une suite de caractères et d'opérateurs :

`" "`	le texte compris entre les guillemets n'est pas interprété
`\`	le caractère suivant est interprété tel quel
`[ ]`	pour définir un ensemble de caractères ; à l'intérieur des crochets, les opérateurs sont ignorés sauf - (pour les définitions d'intervalle), ^ (pour le complémentaire de l'ensemble) et \ (séquence d'échappement ordinaire)
`?`	élément facultatif
`.`	n'importe quel caractère (sauf la fin de ligne)
`*`	0 ou plusieurs fois
`+`	1 ou plusieurs fois
`|`	alternative
`( )`	pour grouper
...	...

Les actions

Lorsqu'un mot est identifié, l'action par défaut est celle qui consiste à copier l'entrée standard sur la sortie standard.

`yytext`	tableau de caractères contenant la chaîne reconnue pour l'expression régulière en cours
`yyleng`	nombre de caractères de la chaîne reconnue
`yymore()`	indique qu'il faut rajouter la prochaine entrée reconnue à cette entrée
`yyless(n)`	n caractères à retenir de l'entrée courante
`input()`	prochain caractère lu
`output(c)`	écrit le caractère c sur la sortie
`unput(c)`	retourne le caractère c sur le flot d'entrée

Exercice 01

Soit le code Flex suivant :

```
%%
[ \t]+ putchar( '_' );
[ \t]+$             /* ignorer */
```

Après voir le résultat de l'exécution. Décrire la fonction de ce code.

Exercice 02

Soit le code Flex suivant :

```
%%
a |
ab |
abc |
UFAS ECHO; REJECT;
.|\n  /* ignorer les caractères non reconnus */
```

Après voir le résultat de l'exécution. Décrire le fonctionnement de ce code.

Exercice 03

Soit le code Flex suivant :

```
%%
algerie ECHO; yyless(3);
[a-z]+ ECHO;
```

Que donne ce code ?

Exercice 04

Soit le code Flex suivant :

```
mot L3A
int i = 0;
%%
{mot} i++;
\n ;
. ;
%%
int main (int argc, char *argv[]) {
yylex();
printf("%d\n", i);
return 0;
}
```

qu'imaginez-vous que FLEX génère ?

Exercice 05

Écrire un lexeur flex qui compte (dans fichier) le nombre de :

a. caractères,
b. de mots,
c. des identificateurs et
d. de lignes;

```
chiffre [0-9]
lettre [a-z]|[A-Z]
alphanum {chiffre}|{lettre}
blanc " "
%{
int Nb_Car, Nb_Word, Nb_Id, Nb_Line;
%}
%%
({alphanum})+ {ECHO; Nb_Car += yyleng; Nb_Word++;}
({blanc})+ {ECHO; Nb_Car += yyleng;}
{lettre}({chiffre}|{lettre})* {ECHO; Nb_Id ++;}
\n {ECHO; Nb_Line++;}
```

```
%%

#include <stdio.h>
int main()
{
Nb_Car=Nb_Word=Nb_Id=Nb_Line=0;
yylex() ; /* pas de return */
printf("\nNb_Car=%d\nNb_Word=%d\nNb_Id=%d\nNb_Line=%d\n\n",Nb_Car,Nb_Word,NB_Id,Nb_Line);
return(1);
}
```

Exercice 06

Écrire un lexeur flex qui ajoute des numéros aux lignes d'un fichier (sauf les lignes blanches) ;

```
blanc " "|\t
%{
#define yywrap() 1 /* GNU flex */
int Nb_Line = 1;
%}
%%
^({blanc})*      {}
\n               {}
^.*              {printf("[%d] %s\n",Nb_Line, yytext) ; Nb_Line++;}
%%
#include <stdio.h>
int main()
{ yylex(); return(1); }
```

Exercice 07

Écrire un lexeur flex qui n'affiche que les commentaires d'un programme. Ceux-ci sont compris entre {};

```
%%
"{"([^{}])+"}" {ECHO;}
.|\n {}
%%
#include <stdio.h>
int main()
{
yylex();
return(1);
}
```

Exercice 08

Écrire un lexeur flex qui remplace dans un texte le mot groupe par section si la ligne début par a, par classe si la ligne débute par b.

```
%{
int i;
%}
%%
\n {ECHO; i=0;}
^a {ECHO; i=1;}
^b {ECHO; i=2;}
groupe {if(i==1) printf("section");
       if(i==2) printf("classe");
       if(i==0) printf("groupe");}
. {ECHO;}
%%
#include <stdio.h>
int main()
{
    i=0;
    yylex();
    return(1);
}
```

Exercice 09

Écrire une fonction d'analyse lexicale à l'aide de LEX. Les *tokens* reconnus seront :

- Les nombres décimaux (en notation scientifique) ;
- Les identifiants de variables ;
- Les opérateurs relationnels RELOP (**< : PPQ, > :PGQ, <= : PPE, >= :PGE ,<> :DI**F) ;
- Les mots-clefs **si**, **sinon** et **alors**.

En utilisant les fonctions suivantes :

- RangerID : range l'identificateur dans la TS.
- RangerId() : // numéro

Cette fonction a pour but d'être utilisée dans un analyseur syntaxique écrit en YACC.

```
%{
#define PPQ 1
#define PPE 2
/* etc... */
%}
/*Definition regulieres*/
delim [ \t\n]
bl {delim}+
lettre [A-Za-z]
chiffre [0-9]
id {lettre}+({lettre}|{chiffre})*
nombre {chiffre}+(\.{chiffre}+)?(E[+-]?{chiffre}+)?
%%
{bl} {/* Pas d'action, pas de retour */}
si {return(SI);}
alors {return(ALORS);}
sinon {return(SINON);}
{id} {yylval=RangerId(); return(ID);}
{nombre}{yylval=RangerNB(); return(NOMBRE);}
"<" {yylval=PPQ; return(OPREL);}
"<=" {yylval=PPE; return(OPREL);}
"=" {yylval=EGA; return(OPREL);}
"<>" {yylval=DIF; return(OPREL);}
">" {yylval=PGQ; return(OPREL);}
">=" {yylval=PGE; return(OPREL);}
%%
RangerId() {
/*
Ranger dans la table des symboles le lexeme yytext
*/
}
RangerNb() {
/*Idem pour un nombre (Conversion)*/
}
```

Exercice 10 :

Ecrire un lexeur flex capable de reconnaitre les entités suivantes :

– Les mot-clefs : **begin**, **end** ;

– les identificateurs (séquences de chiffres et de lettres commençant par une lettre, et qui ne sont pas des mot-clefs) ;

– les opérateurs : +, –, * et **.

Les espaces, tabulations et retours à la ligne sont des séparateurs. Toute autre séquence de caractères qui ne forme pas une entité est considérée comme une erreur.

Le lexeur affichera à l'écran la liste des entités reconnues : "nombre 12", "identificateur abc", "erreur", etc.

Exercice 11:
Ecrire un lexeur flex capable de compter les voyelles, consonnes et caractères de ponctuation d'un texte.

```
%{
int nbVoyelles, nbConsonnes, nbPonct;
%}
consonne [b-df-hj-np-tv-xz]
ponctuation [,;:?!\.]
%%
[aeiouy] nbVoyelles++;
{consonne} nbConsonnes++;
{ponctuation} nbPonct++;
.|\n // ne rien faire
%%
main()
{nbVoyelles = nbConsonnes = nbPonct = 0; yylex();
printf("Il y a %d voyelles, %d consonnes et %d ponctuations.\n",nbVoyelles, nbConsonnes,
nbPonct); }
```

Exercice 12:

Ecrire un lexeur FLEX qui insert le numéro de ligne à chaque ligne dans un fichier.

```
%{ int yyNumLigne;
%}
%%
^(.*)\n printf("%4d\t%s", ++yyNumLigne, yytext);
%%
int main(int argc, char *argv[])
{
yyin = fopen(argv[1], "r");
yylex();
fclose(yyin);
}
```

Exercice 13:
Ecrire un lexeur FLEX qui reconnaît :

- les nombres Binaires.
- Les nombre du système octale.

```
%%
(0|1)+   printf(‘’ un nombre binaire !\n") ;
```

Une autre version qui n'affiche que les nombres binaires reconnus.

```
%%
(0|1)+   printf("un nombre binaire %s !\n", yytext) ;
```

Systeme octale

```
%%
(0|1|2|3|4|5|6|7)+   printf(‘’ un nombre octale !\n") ;

%%
(0|1|2|3|4|5|6|7)+   printf("un nombre octale %s !\n", yytext) ;
```