Générateurs d'analyseurs

Lex et Yacc

Générateur d'analyseurs lexicaux

I- Lex : (Flex sous linux)

C'est un générateur d'analyseurs lexicaux. Ce générateur produit un programme en langage C qui reconnaît dans un texte source les expressions régulières spécifiées par l'utilisateur. Le programme ainsi engendré porte le nom « lex.yy.c » et contient une fonction « yylex() ».

Lex

ex : programme Lex qui supprime les espaces et tabulations en fin de ligne et remplace toute séquence d'espaces par un seul espace.

```
%%
[ \t]+$      ;
«  »+       printf(«  ») ;
```

Compilation par lex : lex prog. l ➔ génère lex.yy.c

Compilation du fichier généré par lex :
gcc -o executable lex.yy.c -ll

Lex

Un fichier de description pour LEX est constitué de trois parties, selon le schéma suivant :

Définitions

%%
règles qui sont un couple : expressions régulières , actions
%%

fonctions utilisateurs

Expressions Régulières

X	Le caractère x
« x »	Le caractère x même si c'est une opération
\x	Si X est un 'a', 'b', 'f', 'n', 'r', 't' ou 'v', représente l'interprétation ANSI-C de \X
[xy]	x ou y
[x-y]	Un cas dans l'intervalle x..y
[^x]	Sauf x
.	N'importe quel caractère sauf \n
^x	x mais seulement en début de ligne

Expressions Régulières

X$	x mais seulement en fin de ligne
X?	Zéro ou x (c'est-à-dire x est optionnel)
X*	Zéro R ou plus, où R est n'importe quelle expression régulière
x+	Un x ou plus
x.y	x ou y
[x]	Le caractère x
x/y	x si suivi de y

x{m, n} De m à n occurrence de x

Actions Lex

Actions :

L'action par défaut est de recopier l'entrée dans la sortie. (ECHO)

yytext : chaîne de caractères (qui se comporte comme un tableau de caract) qui contient le mot en cours d'analyse ou à analyser (variable prédéfinie)
yytext – yyleng : taille
yywrap() : fonction qui s'exécute à la fin de l'analyse du texte.

Définitions du Lex

partie définitions :

2 types de définitions :

- définitions en langage C contenues entre les 2 symboles :

```
%{
….

 %}
```

- définitions du lex sous forme de couple :

```
nom        chaîne
```

Définitions du Lex

Exemple : programme de reconnaissance de nombres entiers et réels

```
%{
#include <stdio.h>
int ….
%}
C     [0-9]
e     [Ee][-+] ?{c}+
%%
{c}+        printf(« entier ») ;
{c}+ ». »{c}*[{e}] ?
{c}+{e}        printf(« réel ») ;
```


Exercice : Ecrire un programme en Lex qui lit un texte quelconque et crée un histogramme sur les mots

(ex : combien de mots de 5 caractères…) il faut ignorer tous les caractères autres que les lettres

```
%{
#define MAX 30
Int T[MAX] ;  // on déclare  un tableau de MAX
%}
%%
[a-zA-Z]+    T[yyleng]++ ;  //yyleng : longueur du mot courant
. |
\n     ;
%%
yywrap()
{  int i ;
   printf(« longueur des mots ») ;
   for(i=0 ;i<MAX ;i++) if(T[i]>0) printf(« %d mots de longueur %d\n »,T[i],i) ;
}
```

Yacc

yacc est un générateur d'analyseurs syntaxiques .
Le fichier généré par yacc y.tab.c contient la fonction yyparse().

Remarques : on peut utiliser lex seul, yacc seul ou les 2 combinés.

yacc prog.y ➔ y.tab.c
gcc -o exécutable y.tab.c –ly

Yacc

Un fichier de description pour yacc est constitué de trois parties, selon le schéma suivant :

définitions
%%
règles actions
%%
fonctions et procédures

ex :
Pour traduire $A \rightarrow BC$D | EF | G ;
A: BCD
| EF
| G
;

Yacc

Passage de valeurs :
Dans la règle A –> BCD , $$ est la valeur associé à A, $1 associé à B, $2 à C, $3 à D, etc. Le principe d'évaluation est de transférer le résultat de la partie droite de la règle vers la partie gauche.
Exemple :

```
A : B C        { printf (« règle A\n ») ;
                 $$=-1 ;
               }
E : E + T      {$$=$1+$3 ;}
  ;
```

La valeur associée à la règle de grammaire est rangée dans la variable yyval. (variable utilisée par le Lex pour transmettre la valeur au Yacc)

Combinaison de Lex + Yacc

Lex + yacc :

dans la 1ère partie du lex
#include « y.tab.h »

dans la 3ème partie du yacc
#include « lex.yy.c »

compilation :

```
yacc  -d fichier.y          : génère y.tab.h
lex fichier.l
gcc  y.tab.c  -ly  -ll
```

Priorité et précédence

Pour changer les priorité des opérateurs :

%left ‘+’ ‘-‘
%left ‘/’ ‘*’

La priorité augmente en descendant .