Faculté des Sciences & Techniques — Université de Limoges

Licence 3e année

Traitement syntaxique & Sémantique

Corrections – TD n°1

Utilisation de Lex & YACC

Utilisation de Lex

1 – Une première proposition :

```
%{
   #include <stdio.h>
%}
identifiant [a-zA-Z][a-zA-Z0-9]*
%start VALEUR
%%
<INITIAL>{identifiant}   { printf("La variable: %s ", yytext); }
"=="          { printf("a pour valeur");
            BEGIN VALEUR;}
" "           /* ignorer */
<VALEUR>{identifiant}    { printf(" %s\n", yytext);
            BEGIN INITIAL;
          }
%%
```

Deux autres versions moins bonnes :

On peut intégrer le « == » à la reconnaissance de l'identifiant :

```
%{    #include <stdio.h>
%}
identifiant [a-zA-Z][a-zA-Z0-9]*
%%
"Lieu==" { printf("La variable lieu "); }
{identifiant} { printf("a pour valeur %s ",yytext); }
.               /* ignorer */
\n              /* ignorer */
%%
```

On peut utiliser la fonction « `yyless` » qui permet de manipuler le « pointeur » d'analyse du moteur interne de Lex : on lit les caractères « == » pour la reconnaissance de l'identifiant, puis on recule le pointeur pour remettre ces caractères dans le flux à analyser.

```
%{ #include <stdio.h>
%}
identifiant [a-zA-Z][a-zA-Z0-9]*
%%
{identifiant}== { char buffer[256];
                  int i;
                  for(i=0; (i < yyleng-2) &&
                                    (i <256);i++)
                     buffer[i]= yytext[i];
                  buffer[i]='\0';
                  printf("La variable: %s ", buffer);
                  yyless(yyleng-2);
                  }
"==" { printf("a pour valeur"); }
" " /* ignorer */
{identifiant} { printf(" %s\n", yytext); }
%%
```

2 – En testant s'il est multiple de 2 :

```
%{
   #include <stdio.h>
   int compteur = 0;
%}
entier ([1-9][0-9]*)|0
nonentier 0[0-9]+
%start IGNORER
%%
<INITIAL>{entier} { int mon_entier = atoi(yytext);
      if ((mon_entier % 2) == 0)
      { compteur++;
         printf("<%d>",mon_entier);
      }
      if (compteur == 10) /* exit(0); */
      BEGIN IGNORER;
   }
<INITIAL>{nonentier} /* ignorer */
<IGNORER>.*   /* ignorer */
%%
```

3 – On veut obtenir les statistiques sur un texte :

```
%{
   #include <stdlib.h>
   int nb_mots = 0;
   int nb_lignes = 0;
   int nb_caracteres = 0;
   float moyenne = 0.0;
%}
mot [[:alnum:]\-_]+
%start POUREOF
%%
{mot}   {
      nb_mots++;
      nb_caracteres += yyleng;
      printf ("[%d]",yyleng);
      // ECHO;
      BEGIN POUREOF;
   }
\n   { nb_lignes++;
     BEGIN INITIAL;
   }
<POUREOF><<EOF>>        {
                              nb_lignes++;
                              yyterminate();
                        }
.   /* ignorer */
%%
void main()
{
   yylex();
   if (nb_mots != 0)
      moyenne = nb_caracteres/nb_mots;
   printf("La moyenne de la taille des mots est %.2f\n", moyenne);
   printf("Le nombre de ligne est %d et de mot %d\n", nb_lignes,nb_mots);
}
```

Une deuxième version :

```
%{
   #include <stdlib.h>
   int nb_mots = 0;
   int nb_lignes = 0;
   int nb_caracteres = 0;
   float moyenne = 0.0;
   void afficher_stats();
%}
mot [[:alnum:]\-_]+
%start POUREOF
%%
{mot}   {
      nb_mots++;
      nb_caracteres += yyleng;
      printf ("[%d]",yyleng);
      // ECHO;
      BEGIN POUREOF;
   }
\n   { nb_lignes++;
     BEGIN INITIAL;
   }
<POUREOF><<EOF>>   {
             nb_lignes++;
            afficher_stats();
            yyterminate();
         }
<<EOF>>   {
      afficher_stats();
      yyterminate();
   }
.   /* ignorer */
%%
void afficher_stats()
{
   if (nb_mots != 0)
      moyenne = nb_caracteres/nb_mots;
   printf("La moyenne de la taille des mots est %.2f\n", moyenne);
   printf("Le nombre de ligne est %d et de mot %d\n", nb_lignes,nb_mots);
}
```

4 – À la recherche du texte perdu :

```
%{
   #include <stdlib.h>
   int nb_sauts = 0;
%}
mot [[:alnum:]\-_]+
%%
{mot}   {
      if (nb_sauts == 0)
         {
           ECHO; // affiche yytext
           printf(" ");
           nb_sauts = yyleng - 1;
         }
      else
         nb_sauts--;

   }
.   /* ignorer */
%%
```

5 – Les attributs des balises :

```
%{
   #include <stdio.h>
%}
identifiant [a-zA-Z][a-zA-Z0-9]*
%start CHAINE CHAINEVIDE BALISE AFFECTATION BALISEDEBUT PRECHAINE COMMENTAIRE
%%
<INITIAL>"<" { BEGIN BALISEDEBUT; }
<INITIAL>.   /* ignorer */
<INITIAL>"<!--" { BEGIN COMMENTAIRE; }
<COMMENTAIRE>. /* ignorer */
<COMMENTAIRE>"-->" { BEGIN INITIAL; }
<BALISEDEBUT>{identifiant} { printf("Balise :<%s>\n",yytext); BEGIN BALISE;}
<BALISEDEBUT>. { printf("Erreur\n"); yyterminate();}
<BALISE>{identifiant}   { printf("Attribut :<%s>\n",yytext); BEGIN AFFECTATION;}
<BALISE>" "   /* ignorer */
<BALISE>">"   { BEGIN INITIAL; }
<BALISE>. { printf("Erreur\n"); yyterminate();}
<AFFECTATION>=   { BEGIN PRECHAINE; }
<AFFECTATION>" " /* ignorer */
<AFFECTATION>.   { printf("Erreur\n"); yyterminate();}
<PRECHAINE>\"      { BEGIN CHAINEVIDE;}
<PRECHAINE>" "   /* ignorer */
<PRECHAINE>.   { printf("Erreur\n"); yyterminate();}
<CHAINEVIDE>[^\"]*   { printf("chaine:<%s>\n",yytext); BEGIN CHAINE;}
<CHAINEVIDE>\"   { printf ("chaine vide\n"); BEGIN BALISE; }
<CHAINE>\"   {BEGIN BALISE;}
\n   /* ignorer */
%%
```

6 – Cryptanalyse par analyse fréquentielle sur un texte chiffré par un code par substitution.

```
%{
   #include "stdio.h"
   int caractere[256];
   int digramme[256][256];
   int trigramme[256][256][256];
%}
lettre [A-Za-z]
%%
{lettre}{3}   { printf("trigramme:<%s>\n",yytext); REJECT; }
{lettre}{2}   { printf("digramme:<%s>\n",yytext); REJECT;}
{lettre}   { printf("caractere:<%s>\n",yytext); }
%%
.            ;
\n           ;
%%
int main(void)
{
   int i,j,k;
   for(i=0; i< 256; i++)
      caractere[i] = 0;

   for(i=0; i< 256; i++)
           for(j=0; j < 256; j++)
         digramme[i][j] = 0;
```

```
   for(i=0; i< 256; i++)
           for(j=0; j < 256; j++)
         for(k=0; k< 256; k++)
            trigramme[i][j][k] = 0;
   yylex();
   printf("Caracteres :\n");
   for(i=0; i< 256; i++)
      if (caractere[i] != 0)
         printf("%c = %d,",i,caractere[i]);
   printf("\ndigramme :\n");
   for(i=0; i< 256; i++)
           for(j=0; j < 256; j++)
                   if (digramme[i][j] != 0)
                      printf ("%c%c = %d,", i, j, digramme[i][j]);
   printf("\ntrigramme :\n");
for(i=0; i< 256; i++)
        for(j=0; j < 256; j++)
      for(k=0; k< 256; k++)
                if (trigramme[i][j][k] != 0)
                        printf ("%c%c%c = %d,", i, j, k, trigramme[i][j][k]);
}
```

7 – Gestion de compte :

a. Première version :

```
%{
   #include <stdio.h>

   void convertir(char *, int);
   float compte;
%}
VALEUR [0-9]+,[0-9]{2}
OPERATION ^[+\-]{VALEUR}
SEPARATEUR ----
%%

:{VALEUR}   { convertir(yytext, yyleng);
        compte = atof(&yytext[1]);
        printf("Solde initial : %f\n",compte);
      }
{OPERATION}   { convertir(yytext, yyleng);
        compte += atof(yytext);
        ECHO;
      }
{SEPARATEUR}   { ECHO;
        printf("\n= %.2f", compte);
      }

%%
void convertir(char *chaine, int taille)
{
   int i = 0;
   for(i=0; i<taille;i++)
      if (chaine[i] == ',') { chaine[i] = '.';
               break; }
}
```

Ce qui donne :

```
pef@cerberus:~/Compilation2$ ./exo7 < operations.txt
Solde initial : 1500.00


+345.00 # virement compte epargne
-450.00 # reparation voiture
-80.50
+0.509  # +2345,00
--------
= 1315.00
-25.80 # repas
-78.60 # livres
-67.90 # Mass Effect 3 pour Wii
--------
= 1142.70
----
= 1142.70
```

avec :

```
pef@cerberus:~/Compilation2$ more operations.txt
:1500,00
+345,00 # virement compte epargne
-450,00 # reparation voiture
-80,50
+0,509  # +2345,00
--------
-25,80 # repas
-78,60 # livres
-67,90 # Mass Effect 3 pour Wii
--------
```

b. On peut ajouter la règle suivante :

```
<<EOF>>   { printf("\nSolde courant: %.2f\n", compte); yyterminate(); }
```

À la place de la reconnaissance de la fin du fichier à l'aide du symbole spécial « <<EOF>> », on aurait pu ajouter une définition de la fonction « `yywrap` » qui est appelée à la fin du fichier :

```
#.*\n { printf("%s", yytext);}
. /* Rien */
%%
int yywrap() {
   printf("Valeur finale du compte :%2.2f\n", solde_courant);
   return 1;
}
```