TD/ TP : Analyse syntaxique

Soit le programme suivant écrit dans un langage, auquel nous chercherons à créer un analyseur syntaxique.

Code 1

```
debut
      Var = 1;
      c = Var;
      a=4 ;
fin
```

A. Analyse syntaxique

1- Définir les symboles terminaux du langage et les symboles non-terminaux.
2- Définir un élément de départ
3- Définir les règles de réécriture

B. Préparation de l'environnement de travail

1- Télécharger flex.exe sur le lien http://gnuwin32.sourceforge.net/packages/bison.htm
2- Installer bison dans le répertoire C:\GnuWin32
3- Ajouter le chemin C:\GnuWin32\bin dans la variable d'environnement Path

C. Création d'analyseur syntaxique

Saisir le code suivant dans le fichier `lexique.l`

lexique.l

```
%{
extern int lineNumber;
#include "syntaxeY.h"

%}
%option noyywrap
nbr [0-9]
entier {nbr}+
identif [a-zA-Z_][0-9a-zA-Z_]*
%%
debut       { return DEBUT; }
fin         { return FIN; }
[" "\t]     { /* rien */ }
{entier}    { return ENTIER; }
{identif}   { return IDENTIF; }
"="         { return AFFECT; }
";"         { return PTVIRG; }
"\n"        { ++lineNumber; }
.           { return yytext[0]; }
%%return 1;
}
```

– Lancer l'invite de commande et taper la commande suivante
$ flex -olexiqueL.c lexique.l

– Quels sont les fichiers générés par cette commande ?

Saisir le code suivant dans le fichier `syntaxe.y`

syntaxe.y

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

extern FILE* yyin;    //file pointer by default points to terminal

int yylex(void); // defini dans progL.cpp, utilise par yyparse()
void yyerror(const char * msg);

int lineNumber; // notre compteur de lignes
%}

%token DEBUT FIN // les lexemes que doit fournir yylex()
%token IDENTIF ENTIER AFFECT PTVIRG

%start program // l'axiome de notre grammaire
%%
program : DEBUT listInstr FIN {printf(" sqlt pgme\n");}
;

listInstr : listInstr inst
          | inst
;

inst : IDENTIF AFFECT expr PTVIRG {printf(" instr affect\n");}
;

expr : ENTIER      {printf(" expr entier \n");}
     | IDENTIF     {printf(" expr identif \n");}
;
%%
void yyerror( const char * msg){
      printf("line %d : %s", lineNumber, msg);
}
int main(int argc,char ** argv){
      if(argc>1) yyin=fopen(argv[1],"r"); // check result !!!
      lineNumber=1;
      if(!yyparse())
            printf("Expression correct\n");

      return(0);
}
```

- Lancer l'invite de commande et taper la commande suivante
  $ bison -d -osyntaxeY.c syntaxe.y
- Quels sont les fichiers générés par cette commande ?
- Générer l'exécutable de l'analyseur syntaxique en tapant la commande suivante :
  gcc –o prog lexiqueL.c syntaxeY.c
- Saisir le **code 1** dans un fichier txt puis tester l'analyseur syntaxique.