**Nom & Prénom : OUAKIB Amine**

**TD/ TP : Analyse syntaxique**

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

|  |  |
| --- | --- |
| debut  Var = 1; c = Var; a=4 ;  fin | **Code 1** |
|  |

# A. Analyse syntaxique

1. Définir les symboles terminaux du langage et les symboles non-terminaux.

les symboles terminaux : **var , c , a ,4, 1**

les symboles non-terminaux : **debut , = , ; , fin**

1. Définir un élément de départ

**debut**

1. Définir les règles de réécriture

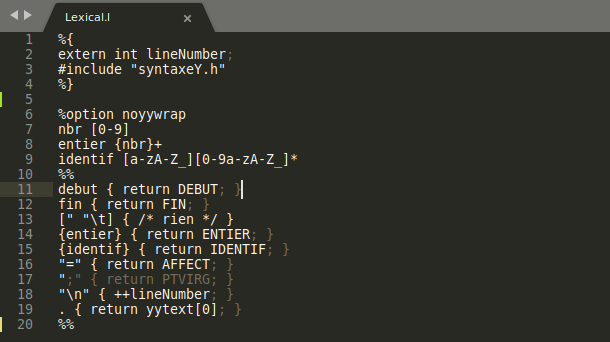
**debut**

**//des instructions**

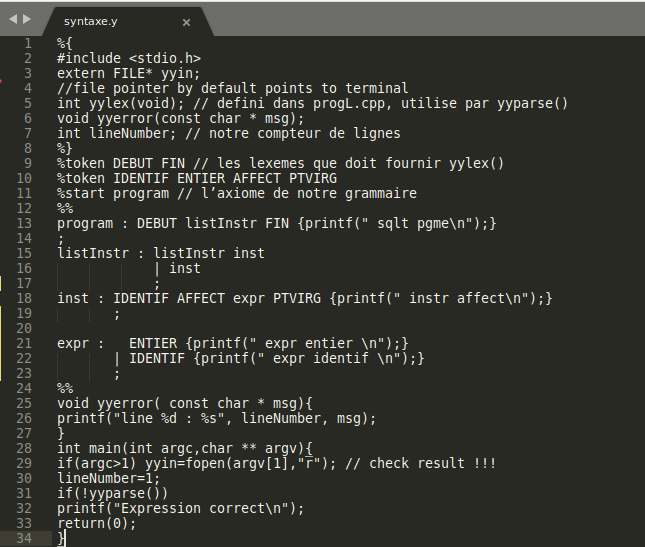
**fin**

# Création d’analyseur syntaxique

Saisir le code suivant dans le fichier lexique.l



Saisir le code suivant dans le fichier syntaxe.y



− Saisir le **code 1** dans un fichier txt puis tester l’analyseur syntaxique.



# − Quels sont les fichiers générés par la commande flex -olexiqueL.c lexique.l

? le fichier générés par la commande précédent est **lexiqueL.c**

# − Quels sont les fichiers générés par la commande bison -d -osyntaxeY.c syntaxe.y

? le fichier générés par la commande précédent est **syntaxeY.c**

− Lancer l’invite de commande et taper la commande suivante

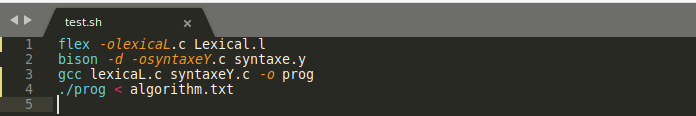
# $ flex -olexiqueL.c lexique.l

− Lancer l’invite de commande et taper la commande suivante

# $ bison -d -osyntaxeY.c syntaxe.y

− Générer l’exécutable de l’analyseur syntaxique en tapant la commande suivante :

# gcc –o prog lexiqueL.c syntaxeY.c



Le résultat obtenant d’après l’exécution :



**Conclusion** : d’après ce travail pratique, on a connu l’intérêt de l’analyse syntaxique pour construire un compilateur, avec cette analyse on peut faire une grammaire de mon langage.