Université Cadi Ayyad Marrakech
Faculté des sciences du travail
Département informatique

ZITOUNI HAMZA

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.
les symboles terminaux : **var , c , a ,4, 1**
les symboles non-terminaux : **debut , = , ; , fin**
2- Définir un élément de départ
debut
3- Définir les règles de réécriture
debut
//des instructions
fin

C. Création d'analyseur syntaxique

- Fichiers syntaxe.y et lexique.l

syntaxe.y lexique.l

D: > s6 > compilation > tp2 > 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; }
pour { return POUR;}
allant_de { return ALLANT_DE;}
jusqua {return JUSQUA; }
[" "\t] { /* rien */ }
{entier} { return ENTIER; }
{identif} { return IDENTIF; }
"="     { return AFFECT; }
";"     { return PTVIRG;}
"\n"    { ++lineNumber; }
.       { return yytext[0];}
%%
```

syntaxe.y lexique.l

D: > s6 > compilation > tp2 > syntaxe.y

```
%%
program : DEBUT listInstr FIN {printf(" sqlt pgme \n");}
;
listInstr : listInstr inst
| inst
;
inst : IDENTIF AFFECT expr PTVIRG {printf(" instr affect \n");}
    |POUR IDENTIF ALLANT_DE ENTIER JUSQUA ENTIER inst PTVIRG {printf("boucle POUR")}
;
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);
}
```

ZITOUNI HAMZA

TP ANALYSE SYNTAXIQUE

- Le fichier text

code.txt

D: > s6 > compilation > tp2 > code.txt

```
1   debut
2   Var = 1;
3   c = Var;
4   a=4 ;
5   pour i allant_de 0 A 10;
6   fin
```

- Les fichiers générés par les deux commandes

flex -olexiqueL.c lexique.l : génère le fichier **lexiqueL.c**
bison -d -osyntaxeY.c syntaxe.y : génère le fichier **syntaxeY.c**

- Test de l'analyseur syntaxique.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.1379]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\s6\compilation\tp2>prog<code.txt
 expr entier
 instr affect
 expr identif
 instr affect
 expr entier
 instr affect
line 5 : syntax error
D:\s6\compilation\tp2>
```

Conclusion :

L'analyse lexicale nous permit de définir notre propre grammaire à respecter par l'utilisateur, et aussi les unités lexicales sont génère automatiquement par bison.