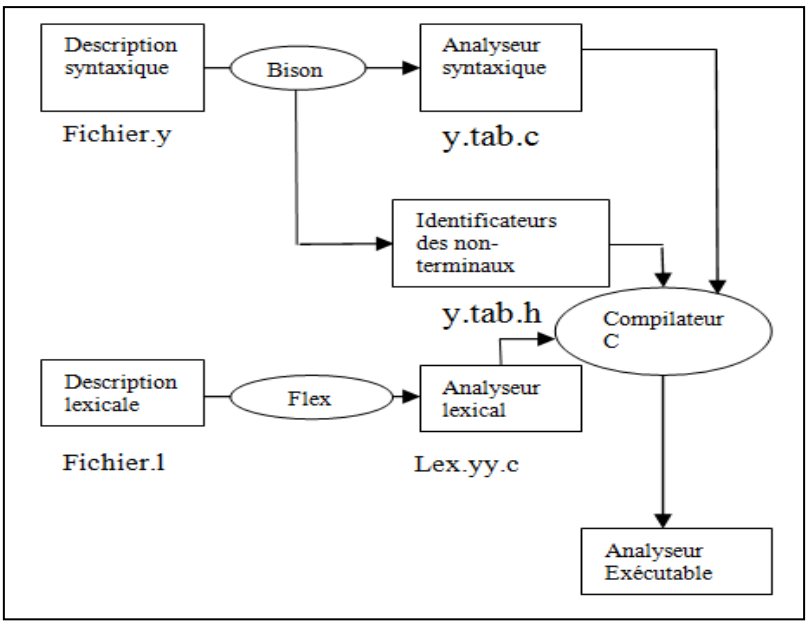
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique
Département d'Informatique

Master 1 :IL

Année Universitaire : 2012/2013
Module : Compile

Manuel de Bison

Bison est un analyseur syntaxique permettant de transformer une grammaire LALR(1) en code C « y.tab.c ». Ce dernier sera compilé afin de générer un code exécutable qui effectue l'analyse syntaxique suivant les instructions données dans le fichier Bison dont l'extension est « .y ». Ce fonctionnement peut être schématisé par la figure suivante:



I. Format d'un fichier bison

Un fichier de spécifications Bison est similaire dans sa structure à celui de Flex. Il se compose de trois parties :

1. La première partie

La première partie d'un fichier Bison peut contenir :

- Les en-têtes, les macros et les autres déclarations C nécessaire à ajouter avant le code de l'analyseur syntaxique.
- La déclaration des symboles terminaux pouvant être rencontrés grâce au mot-clé **%token.**
- Le type de donnée du symbole terminal courant, avec le mot-clé **%union.**

- Des informations donnant la priorité et l'associativité des opérateurs, avec les mots-clé **%left, %right** et **%nonassoc.**
- L'axiome de la grammaire, avec le mot-clé **%start** (si celui-ci n'est pas précisé, l'axiome de la grammaire est la *première* production de la deuxième partie).

```
%{
  Déclaration (en C ) de  variables, constante ,includes ,…
%}
/*Déclarations des unités lexicales utilisées */
  % token  <terminal1> <terminal2> <terminal3>……..
  /*Déclarations de priorités et de types*/
  %left A B C  /*associativité de gauche à droite*/
  %right D E F /* associativité de droite à gauche*/
  %nonassoc  K H /* pas d'associtivité*/
  %start S  /* l'axiome de la grammaire*/
%%
```

Ordre de priorité : - ↓ +

2. **La seconde partie**

Elle décrit la grammaire LALR(1) que l'analyseur doit utiliser. La syntaxe est la suivante :

<Non_terminal1> :<règle de production 1>| <règle de production 2>| ...| <règle de production3>;

« : » : Permet de séparer les parties gauche de la partie droite de la règle.

« | » : Permet de regrouper les règles de dérivation partageant la même partie gauche.

« ; » : Désigne la fin de la régle.

Les actions sémantiques sont des instructions en C insérées dans les règles de productions. Elles sont exécutées chaque fois qu'il y'a une réduction par la production associée.

Exemple :

G : IDF AFF EXPRESSION { /*vérifier la compatibilité des types*/; }

Non terminal — Règle de production — Action sémantique

3. La troisième partie : Bloc principal

La dernière partie contient le code principal de l'analyseur syntaxique ainsi que les définitions des procédures si nécessaire.

```
%{  Déclaration (en C ) de  variables, constante ,includes , …
 %}
 /*Déclarations des unités lexicales utilisées */
 /*Déclarations de priorités et de types*/
%%
  /* la grammaire LALR(1) + les routine sémantiques*/
%%
Int yyerror (char* msg) /* permet d'afficher l'erreur générée*/
{printf(‘’ %s’’,msg)  ;
return 1 ;}
int main()
  { yyparse(); /* permet de lancer l'analyseur syntaxique*/
 return 0;
   }
```

II. Communication avec l'analyseur lexical (Flex):

L'analyseur syntaxique et l'analyseur lexical peuvent communiquer entre eux par l'intermédiaire de la variable « **yylval »**. Cette dernière permet de stocker les entités lexicales récupérées à partir de l'analyseur lexical (Flex) à l'aide de l'instruction ***return***.

La variable ***yylval*** est de type ***YYSTYPE*** (déclaré dans la bibliothèque Bison). Ce dernier est par défaut un **int**. Toutefois, nous pouvons changer ce type par l'instruction suivante :

 # define YYSTYPE autre-type-C ou encore par ***% union { champs d 'une union C }*** qui déclarera automatiquement YYSTYPE comme étant une telle union.

Par exemple :

```
/* Dans la 1er partie du fichier Bison*/
%union {
        int entier ;
        double reel ;
        char * chaine ;
        }
```

ou bien

```
/* Dans la 1er partie du fichier Flex*/
# define  YYSTYPE string
```

La structure de données ***%union*** permet de stocker dans la variable « ***yylval*** » à la fois des ***entiers***, des ***doubles*** et des ***chaînes de caractères***. L'analyseur lexical pourra par exemple contenir :

{nombre} { yylval.entier=atoi(yytext) ; return NOMBRE ; }

Ou bien

{nombre} { yylval.entier=(YYSTYPE)strdup(yytext) ; return NOMBRE ; }

III. Déclaration des types des terminaux et des non terminaux

Le type de ***token*** (terminal) doit être défini à l'aide du nom de champ figurant dans l'union.

Exemple :

% token <entier> NOMBRE

% token <chaine> IDENT CHAINE COMMENT

On peut également définir des types à des non-terminaux.

Exemple :

%type<entier> S

%type<chainr> expr

IV. Compilation

La compilation du fichier bison se fait comme suit :

Flex flex.l

Bison –d bison.y

gcc bison.tab.c lex.yy.c -o nomexécutable

./ nomexécutable

Exemple

Flex .l

```
%{#include<stdio.h>
#include "ts.h" /* pour connaitre les fonctions
insertion, modifier et afficher*/
#include "bison.tab.h" /* pour faire la liaison avec
bison* /
extern YYSTYPE yylval;
%}
chiffre  [0-9]
lettre [a-z]
Nom  {lettre}+
Entier {chiffre}+
grad {lettre}

%%
{nom} {yylval.name=(YYSTYPE)strdup(yytext) ;
          Insérer (nom,tab, 0) ;
           return NOM ;}
{entier}  {yylval.val=(YYSTYPE)strdup(yytext) ;
             return NUM ;}
{grad} {yylval.grade=(YYSTYPE)strdup(yytext) ;
              return GR ;}
%%

Int yywrape()
{return 1 ;
}
```

bison .y

```
%{#include<stdio.h>
extern FILE* yyin ;
extern tab;
%}
%union
{char * name ;
Int num ;
Char grade;}
%token  NOM NUM GR
%start S

%%
S : NOM GR A {modifier($1,$2,tab);} ;/*insérer dans la table
                                          de symbole le
grade*/
A :NUM
   |  /* la production ε est présentée par une production vide */
   ;

%%

Int yyerror(char* msg)
{printf("%s",msg);
return 1;}
Int main()
{
yyin=fopen("entrée.txt",r);
yyparse();
afficher (tab);
return 0;
}
```

Input

Entrée.txt

Ahmed A 20052

output

IDF	Grade
Ahmed	A

Table des symboles