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

TP Compilation

L’analyse syntaxique

Structure d’un fichier BISON

Analyse syntaxique

- L'analyse syntaxique vérifie que les unités lexicales (le résultat de l'analyse lexicale) sont dans le bon ordre défini par le langage.

- **Exemple** : Dans le langage C

  If X==2) y=1;

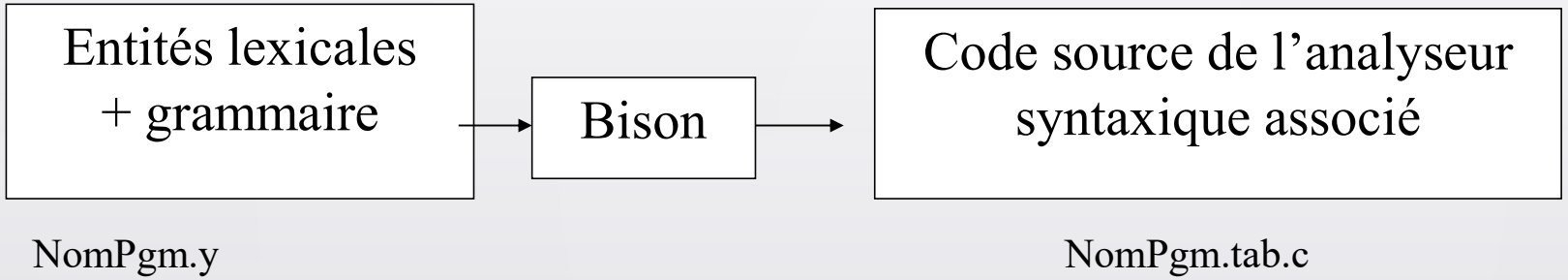
  est une erreur de syntaxe car la règle de if impose une parenthèse avant la condition

- Implémenter un analyseur syntaxique nécessite l'implémentation d'une méthode d'analyse syntaxique vue en cours LL(k), LR(k), SLR(k), LALR(k), ..etc. Ceci nécessite l'écriture de milliers de lignes de code.

Heureusement un outil tel que Bison existe pour nous faciliter la tâche.

L’analyse syntaxique « BISON »

- Bison est un générateur de code d’analyseur syntaxique.
- Il accepte comme entrée la sortie de l’analyse lexicale (les entités lexicales), et la grammaire de langage à analyser (les fichiers bison portent l’extension « .y »).



Structure d’un fichier Bison

Le format d’un fichier Bison est similaire à celui de Flex. Il est composé de trois parties séparées par '%%'.

%{

Définitions en langage C

%}

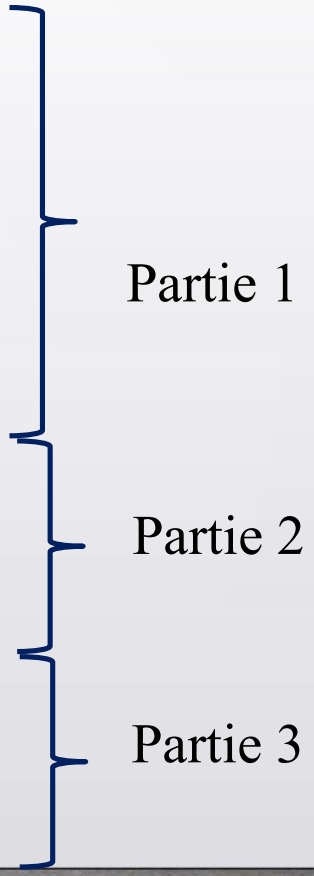
Les définitions des terminaux et d’axiome

%%

Les règles de la grammaire

%%

Redéfinitions des fonctions prédéfinies



Structure d'un fichier Bison

La partie 1:

2.1.Déclaration C (pré-code)

Cette partie peut contenir les en-têtes, les macros et les autres déclarations C nécessaire pour le code de l'analyseur syntaxique.

2.2. Définitions et options

Cette partie contient toutes les déclarations nécessaires pour la grammaire à savoir:

a) **Déclaration des symboles terminaux**

b) **Définition des priorités et d'associativité**

c) **Autres déclarations**

A. Déclaration des symboles terminaux

- ceci est effectué en utilisant le mot clé %token.

Exemple :

%token MAIN IDF Accolade PointVirgule

- On peut préciser le type d'un terminal par

%token<type>.

Exemple :

%token<int> entier

%token<chaine> chaine_cara

B. Définition des priorités et d'associativité

- l'associativité est définie par les mots clé :

%left, %right et %nonassoc.

la priorité est définie selon l'ordre de déclaration des unités lexicales.

Exemple :

%left A B /*associativité de gauche à droite*/

%right C D /* associativité de droite à gauche*/

Ordre de priorité

%nonassoc E F /* pas d'associativité*/

C. Autres déclarations

%start : permet de définir l'axiome de la grammaire.

En l'absence de cette déclaration, Bison considère le premier non-terminal de la grammaire en tant que son axiome.

- **%type :** définir un type à un symbole non-terminal.
- **%union :** permet de spécifier tous les types possibles pour les valeurs sémantiques.

Structure d'un fichier Bison

- **La partie 2:** les règles de production

Ici, on décrit la grammaire LALR(1) du langage à compiler et les routines sémantiques à effectuées selon la syntaxe suivante :

<symbole NonTerminal> : <règle de dérivation 1> {action 1 en langage C }
| <règle de dérivation 2> {action 2 en langage C }
| ...
| <règle de dérivation N> {action N en langage C}
;

Structure d’un fichier Bison

- **Partie 3:** Post-code C

C’est le code principal de l’analyseur syntaxique. Il contient le main ainsi que les définitions des fonctions.

→ Elle doit contenir au minimum les deux fonctions suivantes.

```
main ()
{ yyparse(); }
yywrap ()
{}
```

Lien entre FLEX et BISON

- Afin de lier FLEX à BISON, On doit ajouter dans la partie C du FLEX l'instruction suivante:

```
%{
# include ”NomPgm.tab.h ”
%}
```

Lien entre FLEX et BISON

Exemple

Création de compilateur lexical/syntaxique pour le langage: x=5;

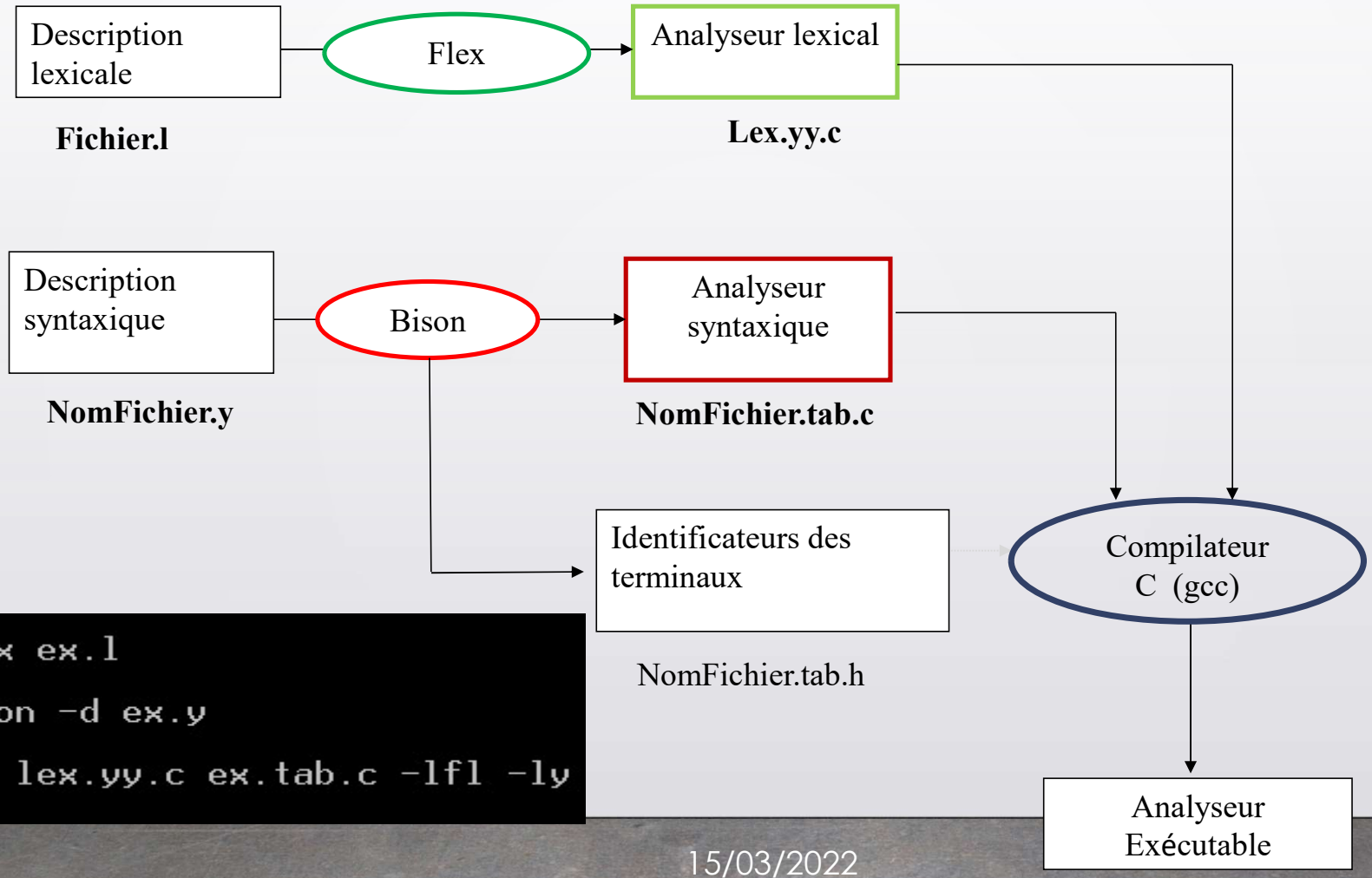
Lexical.l

```
%{
#include “Syntaxique.tab.h"
int nb_ligne=1;
%}
lettre [a-zA-Z]
chiffre [0-9]
IDF  {lettre}({lettre}|{chiffre})*
cst {chiffre}+
%%
{IDF}  return idf;
{cst}    return cst;
=        return aff;
;        return pvg;
[ \t]
\n   {nb_ligne++; }
. printf("erreur lexicale à la ligne %d \n",nb_ligne) ;
```

Syntaxique.y

```
%token idf   cst   aff  pvg
%%
S: idf aff cst pvg {printf("syntaxe correcte");
                    YYACCEPT;}
;
%%
int yyerror(char *msg)
{ printf("%s error syntaxique");
   return 1;  }
main ()
{
yyparse();
}
yywrap()
{}
```

Commande de compilation et d'exécution FLEX/ BISON




```
D:\Exp>flex ex.l
D:\Exp>bison -d ex.y
D:\Exp>gcc lex.yy.c ex.tab.c -lfl -ly
```

Variables et fonctions prédéfinies de Bison

YYACCEPT : instruction qui permet de stopper l'analyseur syntaxique en cas de sucées.

main () : elle doit appeler la fonction yyparse (). L'utilisateur doit écrire son propre main dans la partie du bloc principal.

yyparse () : c'est la fonction principale de l'analyseur syntaxique. On doit faire appelle à cette fonction dans la fonction main().

int yyerror (char* msg) : lorsque une erreur syntaxique est rencontrée, *yyparse* fait appelle à cette fonction. On peut la redéfinir pour donner plus de détails dans le message d'erreur. Par défaut elle est définie comme suit:

```
int yyerror ( char*  msg ) {
printf ( " Erreur Syntaxique rencontrée\n " );
 }
```