ANALYSE SYNTAXIQUE

- L'analyse **syntaxique** vérifie que les entités lexicales (issues de l'analyse lexicale) sont organisées dans l'ordre correct selon les règles du langage.

Exemple: Dans le langage C:

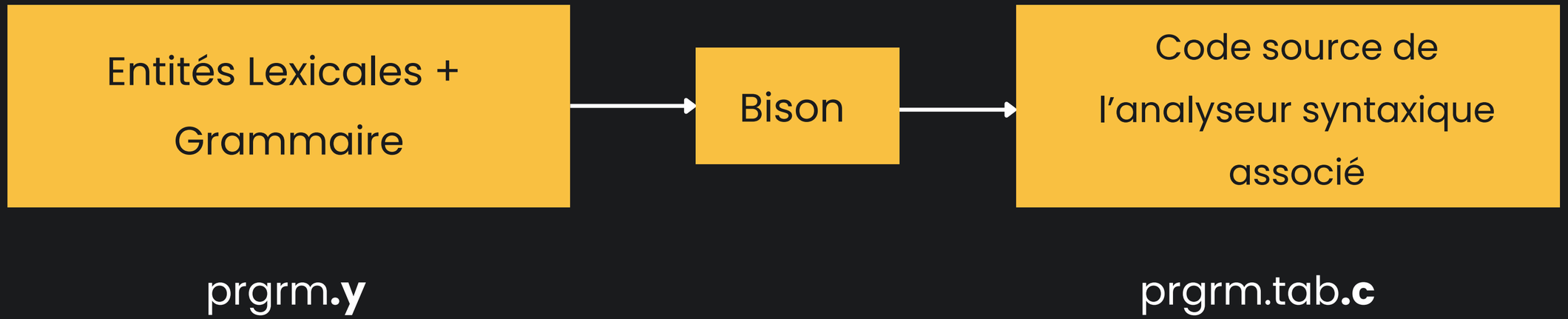
```
#include <stdio.h>
int main()
{  printf("Hello, World!")
   return 0;
}
```

=> **Erreur de syntaxe** car .... ?

- 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

BISON

- Bison est un **générateur** de code d'analyseurs syntaxiques.
- Il prend en entrée les entités lexicales produites par l'analyse lexicale et la grammaire du langage à analyser.
- Les fichiers Bison utilisent l'extension « **.y** ».



FORMAT 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 } Partie 1

%%

Les règles de la grammaire } Partie 2

%%

Redéfinitions des fonctions prédéfinies } Partie 3

PARTIE 1:

1. Déclaration C (pré-code)

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

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

2.a) Déclaration des symboles terminaux

Ceci est effectué en utilisant le mot clé **%token**.

Exemple :

- **%token** MC_if idf pvg cst

On peut aussi spécifier le type d'un terminal par:

%token<type> terminal

Exemple :

- %token**<int>** entier
- %token**<chaine>** idf

2.b) Définition des priorités et d'associativité

- L'**associativité** : On utilise les mots-clés suivants :

  - %left: Spécifie que l'opérateur est **associatif à gauche.**

Exemple: % left '-' **=>** l'expression a - b - c sera interprétée comme (a - b) - c.

  - %right: Spécifie que l'opérateur est **associatif à droite**.

Exemple: % right '=' **=>** L'expression a = b = c sera évaluée comme a = (b = c).

  - %nonassoc: Spécifie que l'opérateur est non-associatif.

Exemple : %nonassoc '<' '>' **=>** une expression comme a < b < c provoquera une erreur de syntaxe, car il est interdit d'enchaîner cet opérateur sans clarification par des parenthèses.

- La **priorité** est définie selon l'ordre de déclaration des unités lexicales.

2.c) Autres déclarations

- **%start :** permet de définir l'axiome (non-terminal initial) 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

```
%union {
int entier;
char* str;
float numvrg;
}
```

```
%token <str>idf
%token <str> mc_entier mc_reel
%token <entier> integer
%token <numvrg> reel
%token <str> divis
```

Exemple Priorité & associativité

On a la déclaration suivante :

```
%right '='
%left '+' '-'
%left '*'  '/'
```

-> Exemple d'entrée:

x = y = p + q * r - s / t

-> Analyse

x = (y = ((p + (q * r)) - (s / t)))

PARTIE 2 : LES RÈGLES DE PRODUCTION

- Dans cette partie, nous décrivons la grammaire LALR(1) du langage à compiler ainsi que les routines sémantiques à exécuter, en suivant 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}
;

Exemple: %token a b

%%

S:aAb {printf("règle de dérivation ");};

PARTIE 3: POST-CODE C

- C'est le code principal de l'analyseur syntaxique. Il contient le main ainsi que les définitions des fonctions.
- Il 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 "Pgm.tab.h "
%}
```

EXEMPLE: CREATION D'UN COMPILATEUR LEXICAL/SYNTAXIQUE POUR : X=5;

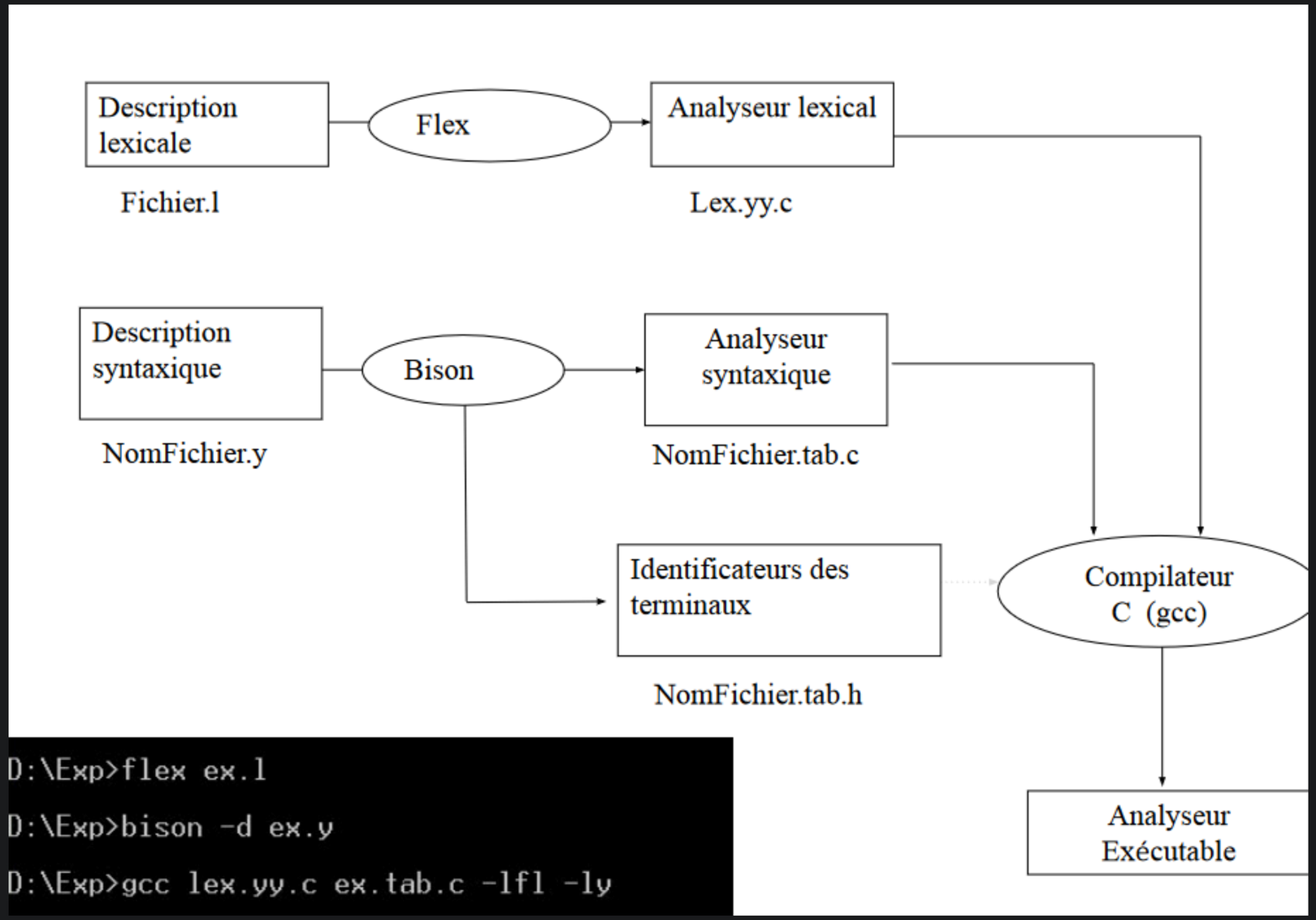
lexical.l

```
%{
#include "syntax.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) ;
%%
```

syntax.y

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

COMMANDES POUR COMPILER ET EXÉCUTER UN PROGRAMME 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 function. 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 à  la ligne %d: \n ",
nb_ligne );
return 1;
}
```