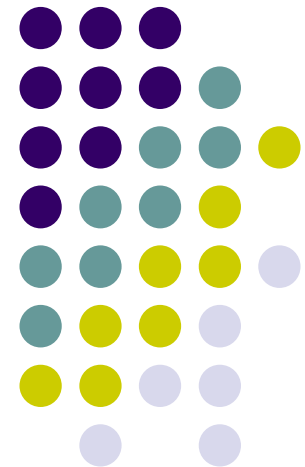
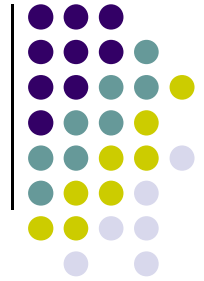


Introduction à YACC

ESI – École nationale Supérieure d'Informatique
Janvier 2014

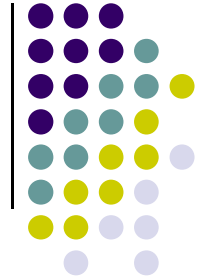


YACC ?

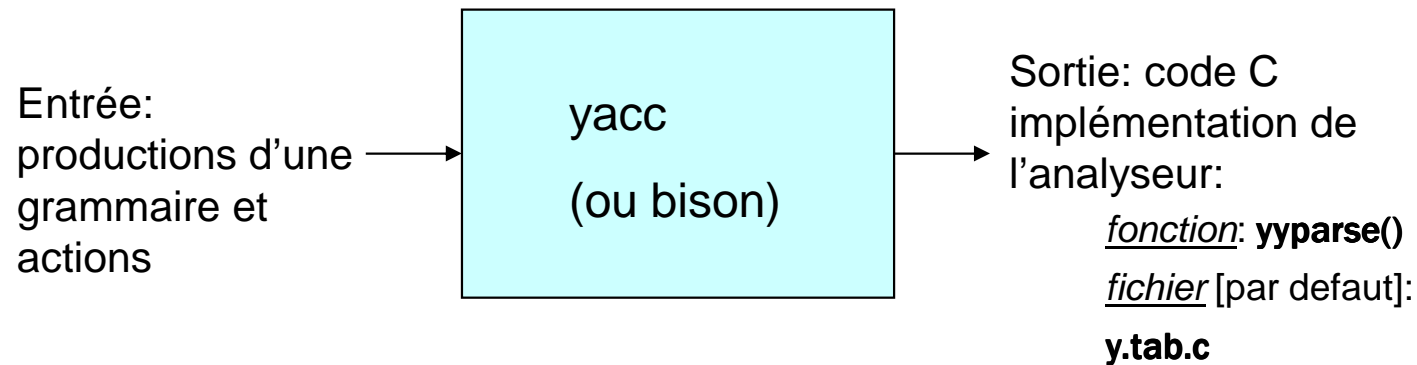


- **Yacc** (**Y**et **A**nother **C**ompiler **C**ompiler) est un programme destiné à compiler une grammaire du type LALR(1) et à produire le texte source d'un analyseur syntaxique du langage engendré par cette grammaire.
- Il est aussi possible, en plus de la vérification de la syntaxe de la grammaire, de lui faire effectuer des actions sémantiques.

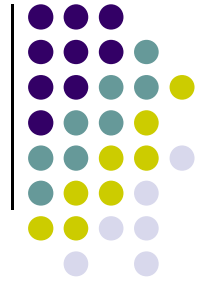
Yacc: Overview



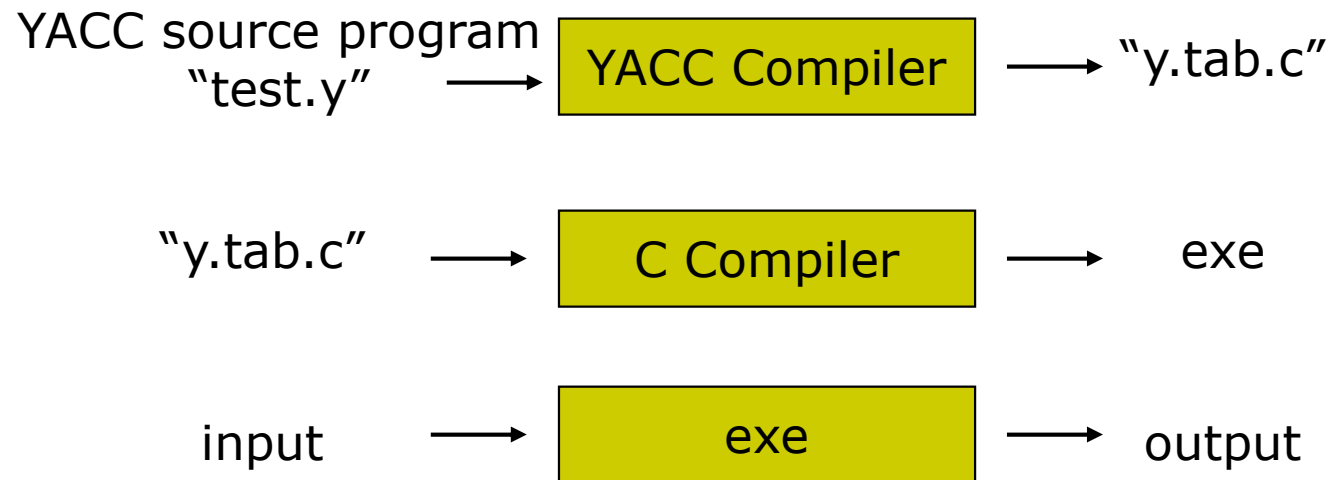
- ▶ En entrée spécifications pour une grammaire de type 2.
- ▶ Produit le code de l'analyseur.



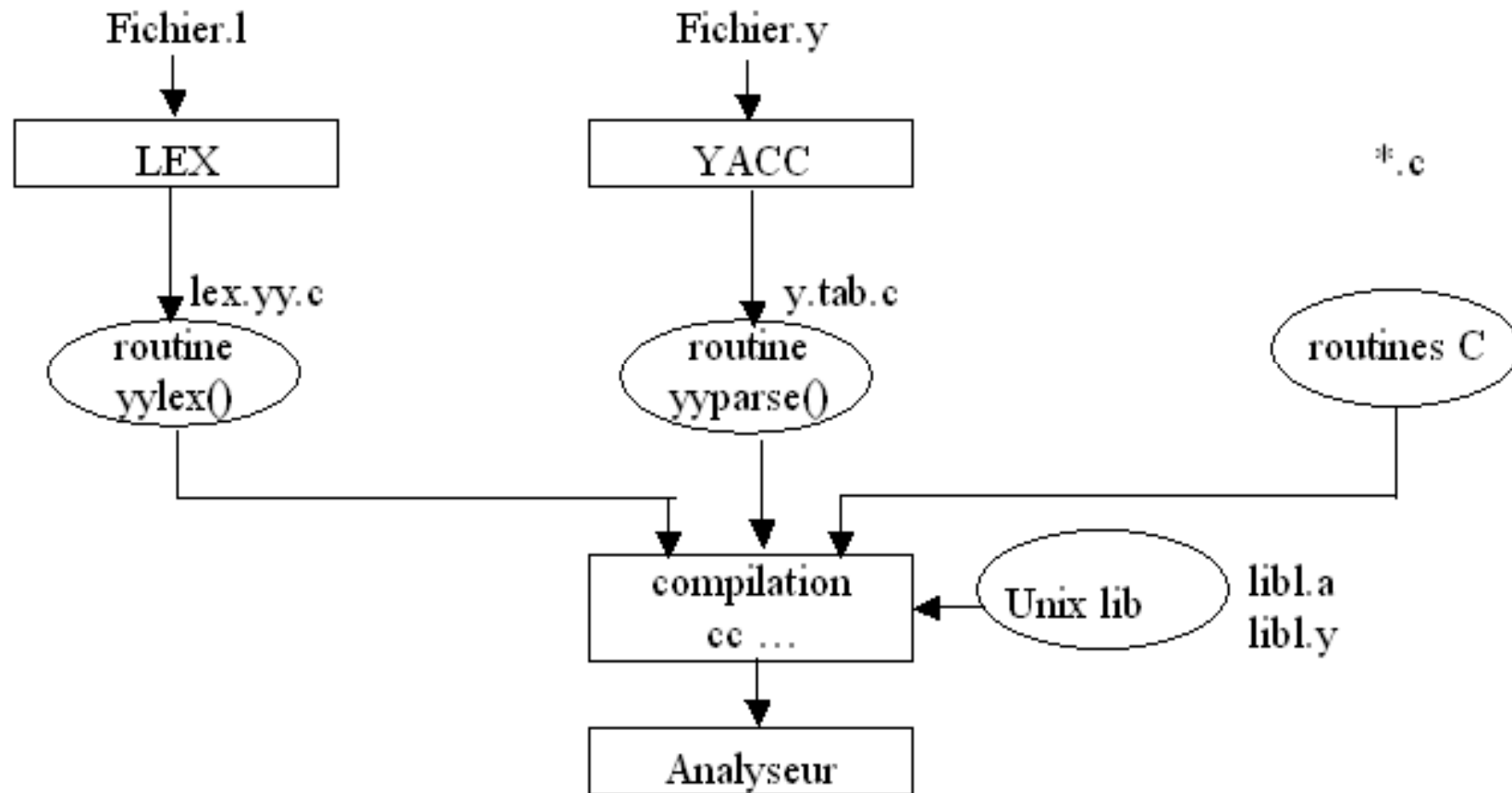
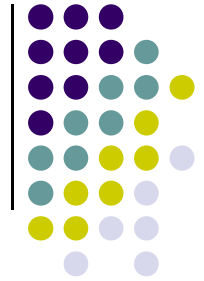
Utilisation



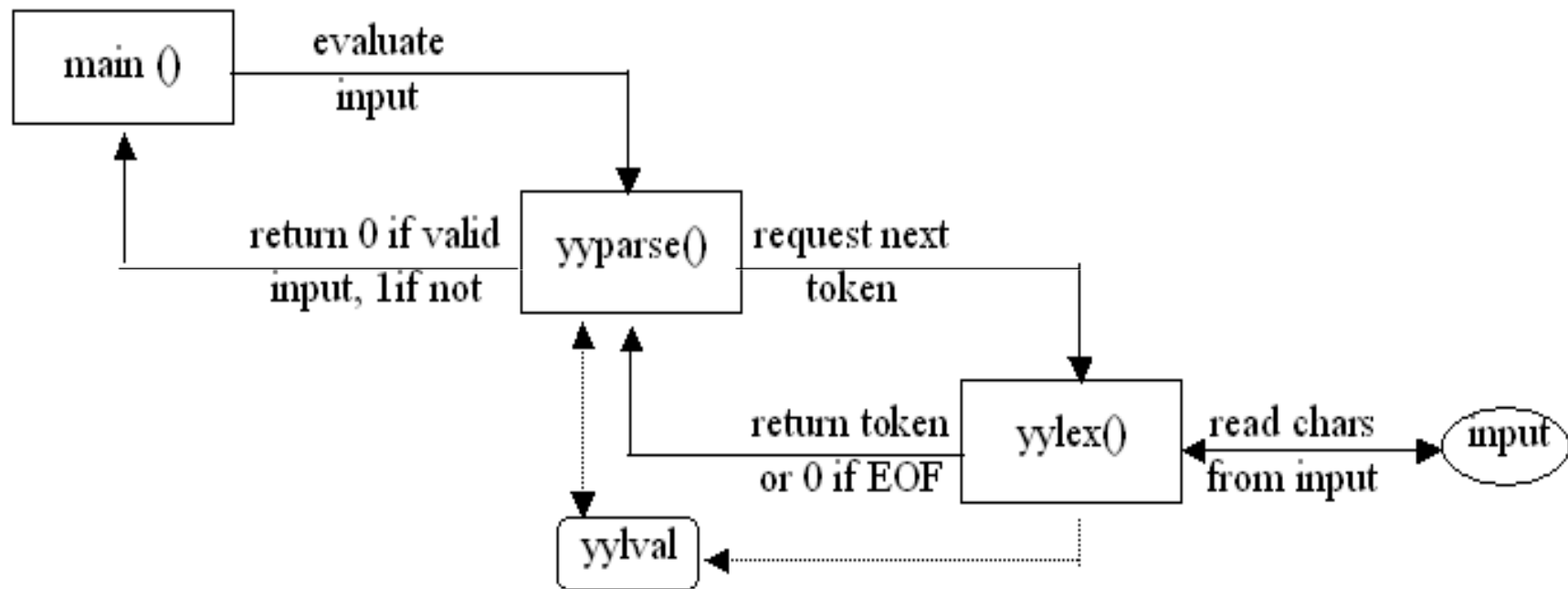
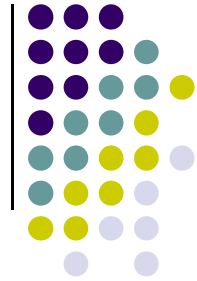
- Générateur YACC



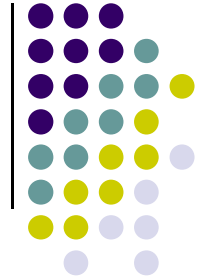
Utilisation



Utilisation

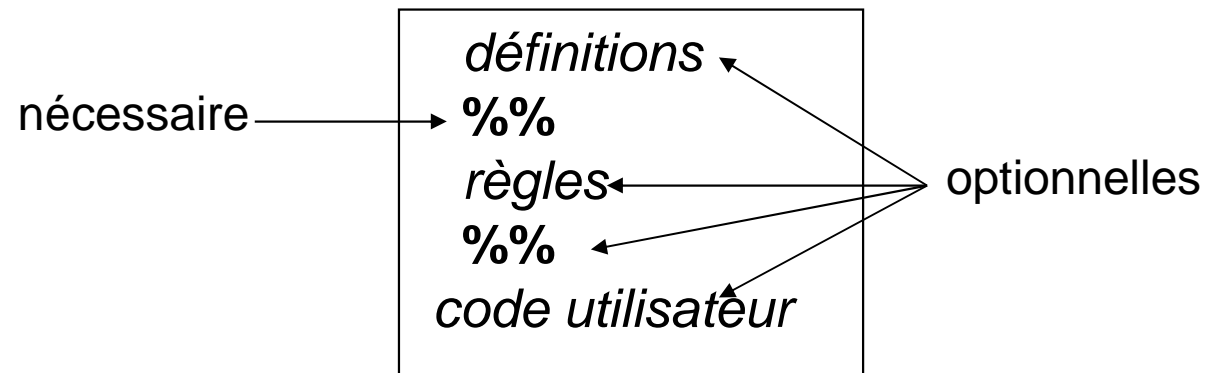


Utilisation



- **Sous Unix**
 - ▶ **\$ lex fichier.l**
 - ▶ **\$ yacc -d fichier.y**
 - ▶ **\$ cc -o analyseur y.tab.c lex.yy.c -ll**

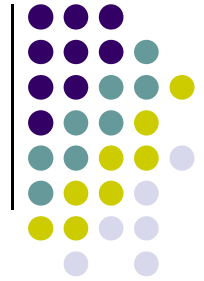
YACC: Structure d'un programme



Plus petite entrée yacc :

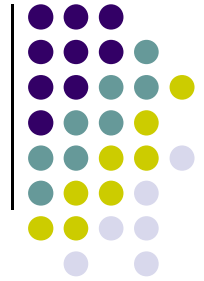
%%

YACC : Partie définitions I



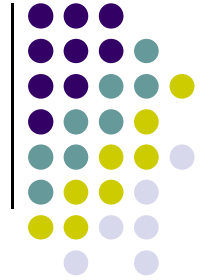
- Peut contenir :
 - ▶ Des spécifications écrites dans le langage cible, placées entre `%{` et `%}`, ces deux symboles étant obligatoirement en début de ligne.
 - ▶ La déclaration des terminaux pouvant être rencontrés, grâce au mot-clé `%token`

YACC : Partie définitions II



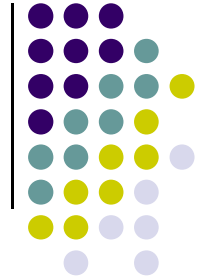
- Peut contenir :
 - ▶ Le type de donnée du terminal courant, avec le mot-clé **%union**
 - ▶ Des informations donnant la priorité et l'associativité des opérateurs.
 - ▶ L'axiome de la grammaire, avec le mot-clé **%start** (si celui-ci n'est pas précisé, l'axiome de la grammaire est le MGP de la *première* production de la deuxième partie).

YACC : Partie règles



- Contient :
 - ▶ Productions de la grammaire et actions sémantiques
 - ▶ Spécifications selon une syntaxe précise
 - ▶ Les actions sémantiques peuvent être au milieu de MDP.

YACC : Partie règles



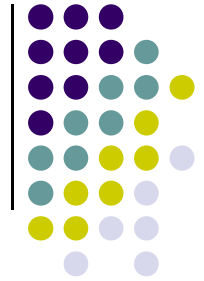
production

$$A \rightarrow B_1 B_2 \dots B_m$$
$$A \rightarrow C_1 C_2 \dots C_n$$
$$A \rightarrow D_1 D_2 \dots D_k$$

Production yacc

$$\begin{aligned} A : & B_1 B_2 \dots B_m \\ & / C_1 C_2 \dots C_n \\ & / D_1 D_2 \dots D_k \\ & ; \end{aligned}$$

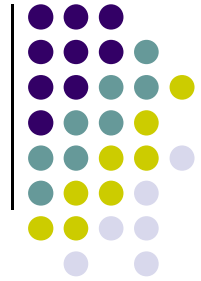
YACC : Partie règles



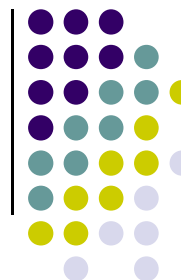
- non_terminal:

```
        corps_1      { action_sémantique_1 }  
    | corps_2      { action_sémantique_2 }  
    | ...  
    | corps_n      { action_sémantique_n }  
    ;
```

YACC : Partie Code Utilisateur



- Code C introduit par l'utilisateur
- Une fonction **main** qui doit appeler **yyparse()**;
int main() { return yyparse(); }
- La fonction **yylex()** est appelée par **yyparse()** ;
doît être définie en utilisant LEX ou en l'écrivant
- fonction **yyerror(char *message)**, appelée
lorsqu'une erreur de syntaxe est trouvée

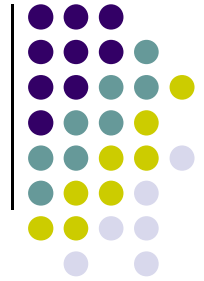


Exemple de programme YACC

```
%{
#include <ctype.h>
%}
%token chiffre

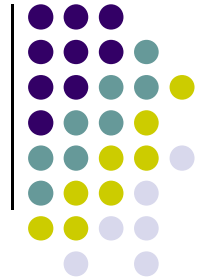
%%
Ligne      :      Expr '\n'          { printf("%d \n", $1); }
Expr       :      Expr '+' Terme     { $$ = $1 + $3; }
| Terme
;
Terme      :      Terme '+' Facteur   { $$ = $1 * $3; }
| Facteur
;
Facteur    :      '(' Expr ')'        { $$ = $2; }
| chiffre
;
%%
main() {yyparse();}
yylex()
{   int c; c = getchar();
    if (isdigit(c)) { yylval = c - '0'; return chiffre ;}
    return (c); }
yyerror() {}
```

Variables et commandes de YACC



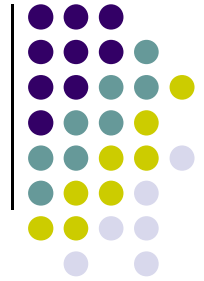
- La production vide est représenté par une alternative vide.
- Les terminaux d'une grammaire spécifiée dans la partie deux d'un programme YACC sont entre ' ' ou les noms déclarés comme entités lexicales.
- Les non terminaux sont les chaînes qui ne sont pas entre ' ' et non déclarés comme entités lexicales.

Variables et commandes de YACC



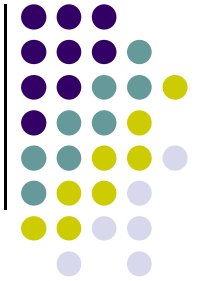
- \$\$ désigne l'attribut associé au MGP (Membre Gauche de Production d'une règle).
- \$i désigne l'attribut associé au i^{ème} symbole d'un MDP (Membre Gauche de Production d'une règle).
- La règle sémantique par défaut est $\{\$ \$ = \$ 1\}$.
- La production vide est représenté par une alternative vide.

Fonctionnement de l'analyseur



- L'analyseur effectue à chaque étape une des quatre actions possibles suivantes :
 - ▶ **shift**
 - ▶ **reduce**
 - ▶ **accept**
 - ▶ **error**
- Si YACC est invoqué avec l'option `-v` (sous Unix) génère un fichier `y.output` qui contient une description lisible de la table d'analyse. Fichier `.v` avec Parser Generator.

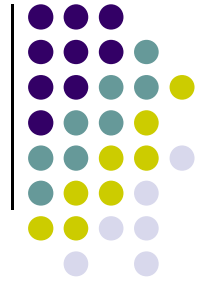
Fonctionnement de l'analyseur



- Exemple de contenu pour le fichier y.output

```
0 $accept : Eval $end
1 Eval : Ligne Eval
2     | '.' '\n'
3 Ligne : Expr '\n'
4 Expr : Expr '+' Terme
5     | Terme
6 Terme : Terme '*' Facteur
7     | Facteur
8 Facteur : '(' Expr ')'
9     | chiffre
```

Fonctionnement de l'analyseur



- Exemple de contenu pour le fichier y.output

state 0

\$accept : . Eval \$end

(' shift 1

'.' shift 2

chiffre shift 3

Eval goto 4

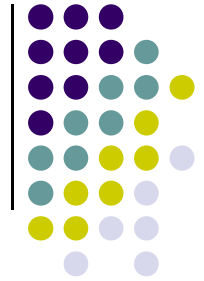
Ligne goto 5

Expr goto 6

Terme goto 7

Facteur goto 8

Gestion des erreurs



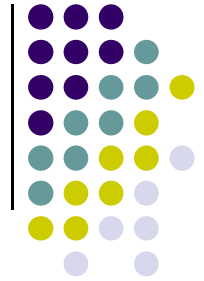
- “token” ‘**error**’ réservé pour la gestion des erreurs :

Exemple:

```
stmt : IF '(' expr ')' stmt  
      | IF '(' error ')' stmt  
      | FOR ...  
      | ...
```

Pour récupération sur erreurs dans ‘expr’

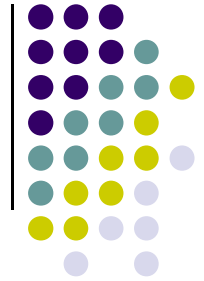
Comportement si erreur



Quand une erreur se produit :

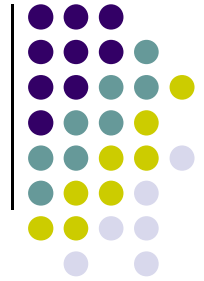
- ▶ Dépiler jusqu'à token 'error' est légale;
- ▶ Se comporte comme si token 'error' vu
 - exécuter action rencontrée;
 - terme courant = celui qui a causé l'erreur
- ▶ Si pas de règles d'erreurs, le calcul se termine.

Conflits



- Des conflits peuvent apparaître si la grammaire en entrée n'est pas LALR(1).
- Deux types de conflit:
 - ▶ *Décalage-Réduction (shift-reduce):*
 - ▶ *Réduction-Réduction (reduce-reduce):*

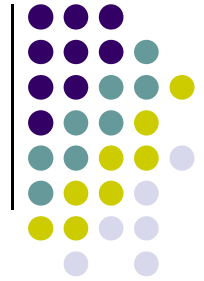
Exemple de conflit Décalage/Réduction



Productions de la Grammaire :

$S \rightarrow \text{if} (e) S \quad /* 1 */$ Entrée: **if** (e_1) **if** (e_2) S_2 **else** S_3
 | **if** (e) S **else** S /* 2 */

Exemple de conflit Réduction/Réduction



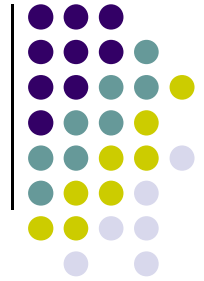
Productions de la Grammaire :

S : A 'a'
 | 'b' A 'c'
 | B 'c'
 | 'b' B 'a'
;

A : 'd'
;

B : 'd'
;

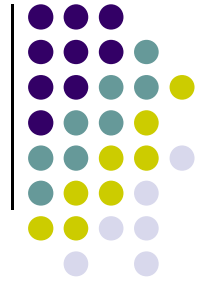
Résolution des conflits dans YACC



- **Priorité des terminaux :**

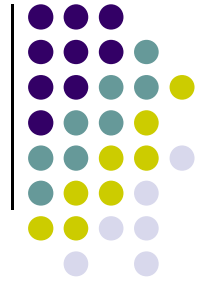
- ▶ Les priorités et associativités sont définies dans la partie déclaration.
- ▶ On spécifie les terminaux des moins prioritaires vers les plus prioritaires.
- ▶ Les terminaux de même priorité sont spécifiés sur la même ligne. Un exemple est donné ci-après :
- ▶ **%left '+', '-'**
- ▶ **%left '*', '/'**
- ▶ **%right moins_unaire**

Résolution des conflits dans YACC



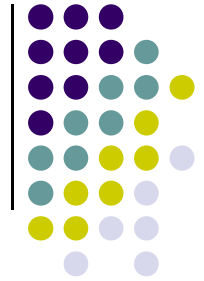
- Priorité d'une règle :
 - ▶ La priorité d'une règle est la priorité du terminal le plus à droite du MDP (membre droit de production).
 - ▶ On peut forcer la priorité d'une règle par la commande : **%prec valeur.**

Résolution des conflits dans YACC



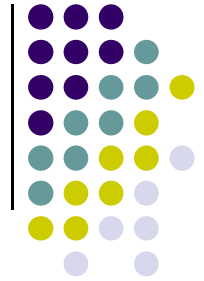
- Si conflit entre Décaler a et Réduire par $A \rightarrow \alpha$, Yacc effectue :
- - **une réduction** si :
 - priorité(règle) > priorité(a) ;
 - même priorité et associativité à gauche.
- -**un décalage** dans tous les autres cas.

Résolution des conflits dans YACC



- **Résolution des conflits Réduire/Réduire :**
 - ▶ Placer la production de réduction préférée en premier dans l'ordre d'apparition des règles.

Résolution des conflits dans YACC - Exemple



%left '+', '-'

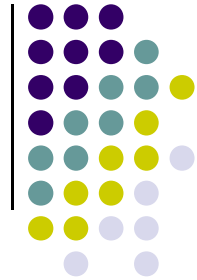
%left '*', '/'

%right moins_unaire

%o%

```
Expr :      Expr '+' Expr  
           | Expr '-' Expr  
           | Expr '*' Expr  
           | Expr '/' Expr  
           | '-' Expr      prec moins_unaire  
  
           ;
```

Exemple y.output



grammaire:

```
S : '0' S '0'
    | /* epsilon */
    ;
```

y.output :

n° production

État en conflit

```
0 $accept : S $end } règle, introduite par yacc
1 S : '0' S '0'      } Règles de la grammaire
2 |
```

...
1: shift/reduce conflict (shift 1, reduce 2) sur '0'

state 1

....

} *information sur l'état 1*