Université Mohamed Khider – Biskra.
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'Informatique

Module: **Compilation**
Niveau : **3ème Licence**
Année universitaire: **2014/2015**
Documents non autorisés

Contrôle

Date : 07/02/2015 **Durée : 1h30m**

Exercice 01 (04 pts)

1. Donner une expression régulière étendue permettant de reconnaître des entiers naturels:
   - Sans zéro non significatif à gauche,
   - Composés de nombre impair de chiffres pairs,
   - ou composés de nombre pair de chiffres impairs.
2. Donner une expression régulière étendue permettant de reconnaître les adresses IP v4, (composé de 4 champs séparés par des points. Chaque champ appartient à l'intervalle [0,255] :
   - La première adresse c'est 0.0.0.0
   - Et l'adresse maximale c'est 255.255.255.255

Exercice 02 (08 pts)

On s'intéresse à une grammaire G des instructions (très simplifiées) délivrées par un logiciel de calcul d'itinéraire, du style avancer 100m, au panneau Biskra tourner à gauche, avancer 20m, tourner à droite. On utilise pour ce faire les symboles GO (avancer Xm), TL (turn left), TR (turn right) et PAN (panneau).

On a G = {VT , VN , route, P} avec VT = { GO, TL, TR, PAN },
VN = { route, inst, panneau, turn } et P contient les productions :

route → inst | route inst
inst → GO | panneau turn
turn → TL | TR
panneau → ε | PAN

1. Cette grammaire n'est pas LL(1) : pourquoi ?
2. Donner une grammaire G1 équivalente à G et qui vous semble LL(1).
3. Donner la table d'analyse LL(1) de G1. Justifier en utilisant cette table que G1 est une grammaire LL(1).
4. Analyser le mot « GO TR TL PAN TL » par un analyseur LL(1).
5. Construire la table SLR de G. G est-elle SLR(1) ? Sinon Justifier rigoureusement en énumérant tous les conflits et en précisant leur types (shift/reduce ou reduce/reduce).

Exercice 03 (08 pts)

Partie I

On souhaite introduire une nouvelle expression arithmétique à 3 alternances avec les propriétés suivantes:

expr_t **:<:** *expr_inf* **:=:** *expr_eg* **:>:** *expr_sup*

Si la valeur de *expr_t* est strictement inférieure à 0 alors la valeur de l'expression est celle de *expr_inf*, sinon si la valeur de *expr_t* est égale à 0 alors la valeur de l'expression est celle de *expr_eg*, sinon la valeur de l'expression est celle de *expr_sup*.

Les opérateurs **:<:** , **:=:** et **:>:** ont une associativité de droite à gauche comme les opérateurs **?:** , mais avec une priorité moindre.

Exemple :

3 – 4 :<: 5 :=: 6 :>: 7 = 5
4 – 4 :<: 5 :=: 6 :>: 7 = 6
4 – 3 :<: 5 :=: 6 :>: 7 = 7

1. Modifier les fichiers Lex et Yacc pour accepter ces opérateurs et pour construire l'arbre correspondant.
2. Donner un modèle de code pour cette nouvelle expression.

Partie II

1. Proposer une fonction récursive (en MNL) permettant de calculer la suite de fibonacci définie de la manière suivante :
   U0 = 0 U1 = 1 Un = Un-1 + Un-2
2. Construire l'arbre correspondant.
3. Ecrire le code MIPS pour cette fonction (selon le principe donné dans le cours).

Fichier Lex

```
.....
%}
ER_ENTIER -?[0-9]+
ER_VARIABLE [a-zA-Z][0-9a-zA-Z]*
ER_SEPARATEUR [ \n\t]
%%
{ER_SEPARATEUR}+ {}
{ER_ENTIER} {yylval.entier = atoi(yytext);
return(TOKEN_ENTIER);}
{ER_VARIABLE} {
yylval.chaine = strcpy((char *)malloc(yyleng+1), yytext);
return(TOKEN_VARIABLE);}
[\+\-] {yylval.caractere = yytext[0];return(TOKEN_PLUS_MOINS);}
[\*\/\%] {yylval.caractere = yytext[0]; return(TOKEN_MULT_DIV);}
[\(\)] {return(yytext[0]);}
"++"|"--" {yylval.caractere = yytext[0];return(TOKEN_INC_DEC);}
```

```
[?:=] {return(yytext[0]);}
<<EOF>> {return(0);}
```

MiniLangage (MNL)	MIPS
lire x	li $2,val syscall sw $2, décalage_de_la_variable($30)
ECRIRE *expression*	*... code de l'expression* *résultat _$8* move $4, $8 *# avec la valeur de l'expression déjà mis dans $8* li $2, 1 syscall
ECRIRE *chaine*	la $4, adresse_de_la_chaine li $2, 4 syscall
SI *expression* **ALORS** *alternance_vraie* **SINON** *alternance_fausse*	*code de l'expression – résultat _$8* bne $8, $9, etiqv *code de l'alternance fausse* j etiqfin etiqv: *code de l'alternance vraie* ... etiqfin: nop
TANTQUE *comparaison* **FAIRE** *corps* **FINTQ**	*j etiqtest* *etiqcorps: code du corps* ... *etiqtest: ...code de l'expression* *résultat_$8* *bne $8, $0, etiqcorps*

Syntaxe	Effet	Syntaxe	Effet
move r1, r2	r1 ← r2	lw r1, o (r2)	r1 → tas (r2 + o)
add r1, r2, o	r1 ← o + r2	sw r1, o (r2)	tas.(r2 + o) ← r1
sub r1, r2, o	r1 ← r2 - o	slt r1, r2, o	r1 ← r2 < o
mul r1, r2, o	r1 ← r2 × o	sle r1, r2, o	r1 ← r2 <= o
div r1, r2, o	r1 ← r2 ÷ o	seq r1, r2, o	r1 ← r2 = o
and r1, r2, o	r1 ← r2 and o	sne r1, r2, o	r1 ← r2!= o
or r1, r2, o	r1 ← r2 or o	j o	pc ← o
xor r1, r2, o	r1 ← r2 xor o	jal o	ra ← pc+1 et pc ← o
sll r1, r2, o	r1 ← r2 sl o	beq r, o, a	pc← a si r = o
srl r1, r2, o	r1 ← r2 sr o	bne r, o, a	pc← a si r <> o
li r1, n	r1 ← n	syscall	appel système
la r1, a	r1 ← a	nop	ne fait rien

Nom	Nº	Effet
print_int	1	imprime l'entier contenu dans a0
print_string	4	imprime la chaîne en a0 jusqu'à '\000'
read_int	5	lit un entier et le place dans v0
sbrk	9	alloue a0 bytes dans le tas, retourne l'adresse du début dans v0.
exit	10	arrêt du programme en cours d'exécution

Fichier yacc

```
%{
…
EXPR_ARBRE a;
%}
%union { EXPR_ARBRE arbre; char *chaine; int entier; char
caractere; }
%token <chaine> TOKEN_VARIABLE
%token <entier> TOKEN_ENTIER
%token <caractere> TOKEN_PLUS_MOINS TOKEN_MULT_DIV
TOKEN_INC_DEC
%token TOKEN_FIN
%type <arbre> EE E T F G V C
%start EE
%%
EE : C {a = $1;};
C : E '?' C ':' C{$$ = CreerTer($1, $3, $5);}
  | E {};
E : E TOKEN_PLUS_MOINS T {$$ = CreerBin($2, $1, $3);}
  | T {};
T : T TOKEN_MULT_DIV G {$$ = CreerBin($2, $1, $3);}
  | G {};
F : '(' C ')' {$$ = $2;}
  | TOKEN_VARIABLE {$$ = CreerVariable($1);}
  | TOKEN_ENTIER {$$ = CreerConstante($1);};
G : TOKEN_INC_DEC V {$$ = CreerUn($1, $2);}
  | F {};
V : '(' TOKEN_VARIABLE ')' {$$ = CreerVariable($2);}
  | TOKEN_VARIABLE {$$ = CreerVariable($1);};
%%
```

Université Mohamed Khider – Biskra.
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'Informatique

Module: **Compilation**
Niveau : **3ème Licence**
Année universitaire: **2014/2015**
Documents non autorisés

Corrigé type

Fait le : 22/01/2014 **Durée : 1h30m**

Exercice 01 (04 pts)

1. 0|[2 4 6 8]([0 2 4 6 8][0 2 4 6 8])*|([1 3 5 7 9][1 3 5 7 9])([1 3 5 7 9] [1 3 5 7 9])* **(02 pts).**
2. (([0-1]?[0-9])?[0-9]|(2[0-4][0-9]|25[0-5])\.){3}(([0-1]?[0-9])?[0-9]| (2[0-4][0-9]|25[0-5]) **(02 pts).**

Exercice 02 (08 pts) : Soit G la grammaire:

route → inst | route inst
inst → GO | panneau turn
turn → TL | TR
panneau → ε | PAN

Cette grammaire n'est pas LL(1) parce qu'il y a une récursivité à gauche dans la règle. **(0,5 pts)**

route → inst | route inst

Donner une grammaire G1 équivalente à G et qui vous semble LL(1).

route → inst route'
route' → inst route'| ε

La grammaire G1 soit comme suit **(1 pts)**:

route → inst route'
route' → inst route'| ε
inst → GO | panneau turn
turn → TL | TR
panneau → ε | PAN

Construire la table d'analyse LL(1) pour G1**(1,5 pts)**.

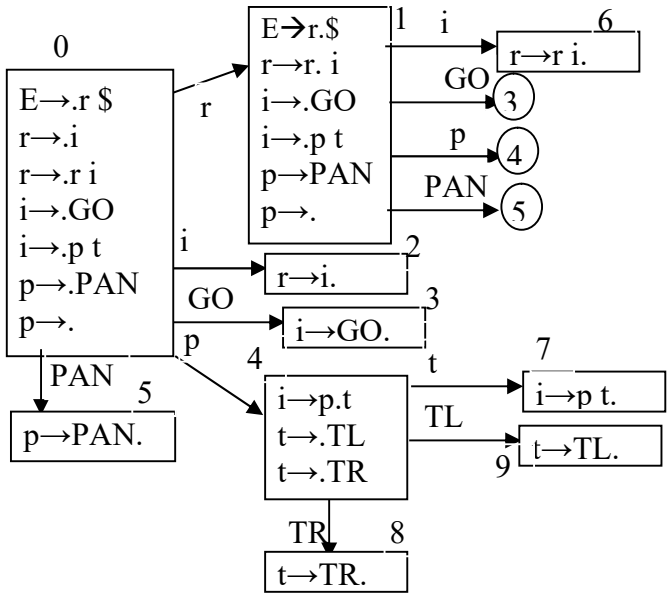
	Premiers	Suivants
Route	GO,PAN,TL,TR	$
route'	GO,PAN,TL,TR, ε	$
Inst	GO,PAN,TL,TR	GO,PAN,TL,TR,$
Turn	TL,TR	GO,PAN,TL,TR,$
Panneau	ε, PAN	TL,TR

	GO	PAN	TL	TR	$
R	r→i r'	r→i r'	r→i r'	r→i r'	
r'	r'→i r'	r'→i r'	r'→i r'	r'→i r'	r'→ ε
I	i → GO	i→ p t	i→ p t	i→ p t	
T			t→ TL	t→ TR	
P		p → PAN	p→ ε	p→ ε	

1. Analyse LL(1) du mot « GO TR TL PAN TL » **(01 pts)**.

Pile	**Chaine**	**Action**
$r	GO TR TL PAN TL $	dépiler, emplier r' i
$r'i	GO TR TL PAN TL $	dépiler, emplier GO
$r'GO	GO TR TL PAN TL $	dépiler, Avancer
$r'	TR TL PAN TL $	dépiler, emplier r' i
$r'i	TR TL PAN TL $	dépiler, emplier t p
$r't p	TR TL PAN TL $	dépiler, emplier ε
$r't	TR TL PAN TL $	dépiler, emplier TR
$r'TR	TR TL PAN TL $	dépiler, Avancer
$r'	TL PAN TL $	dépiler, emplier r' i
$r'i	TL PAN TL $	dépiler, emplier t p
$r't p	TL PAN TL $	dépiler, emplier ε
$r't	TL PAN TL $	dépiler, emplier TL
$r'TL	TL PAN TL $	dépiler, Avancer
$r'	PAN TL $	dépiler, emplier r' i
$r' i	PAN TL $	dépiler, emplier t p
$r't p	PAN TL $	dépiler, emplier PAN
$r't PAN	PAN TL $	dépiler, Avancer
$r't	TL $	dépiler, emplier TL
$r'TL	TL $	dépiler, Avancer
$r'	$	dépiler, emplier ε
$	$	Accépter

Construction de l'automate LR pour G **(2 pts)**.



	Premiers	Suivants
Route	GO,PAN,TL,TR	$
Inst	GO,PAN,TL,TR	GO,PAN,TL,TR,$
Turn	TL,TR	GO,PAN,TL,TR ,$
Panneau	ε, PAN	TL,TR

SLR(1) ? oui il n'y a aucun cas de conflit **(0,5 pts)**

SLR(1)	GO	PAN	TL	TR	$
0	d3	d5	r8	r8	
1	d3	d5	r8	r8	ACC
2	r1	r1	r1	r1	r1
3					r3
4			9	8	
5			r7	r7	
6					r2
7	r4	r4	r4	r4	r4
8	r5	r5	r5	r5	r5
9	r6	r6	r6	r6	r6

(01,5 pts)

r	i	t	p
1	2		4
	6		4
		7	

Exercice 0 3 (08 pts)

Partie 01 :

Fichier Lex (2 pts)

```
……
%}
ER_ENTIER -?[0-9]+
……
%%
{ER_SEPARATEUR}+ {}
……
":<:"{ yylval.chaine = strcpy((char *) malloc(yyleng+1),
yytext);return(TOKEN_OPINF);}
":=:"{ yylval.chaine = strcpy((char *) malloc(yyleng+1),
yytext);return(TOKEN_OPEG);}
":>:"{ yylval.chaine = strcpy((char *) malloc(yyleng+1),
yytext);return(TOKEN_OPSUP);}
```

Fichier yacc

```
%{
…
%token <chaine> TOKEN_VARIABLE TOKEN_OPINF TOKEN_OPEG
TOKEN_OPSUP
……
%start EE
%%
EE : M {a = $1;};
M: C TOKEN_OPINF M TOKEN_OPEG M TOKEN_OPSUP{$$ =
CreerQuad($1, $3, $5,$7);}
|C{}
C : E '?' C ':' C{$$ = CreerTer($1, $3, $5);}
  | E {};
```

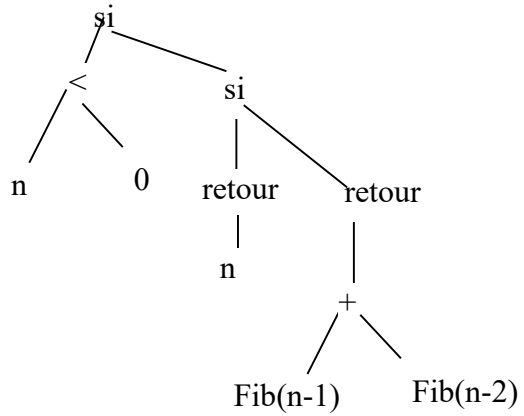
Modèle de code (d'instructions) MIPS pour cet opérateur. **(2 pts)**

```
Code calculant l'expression C →$n
0 →$n+1
BLT $n,$n+1,et1
BEQ $n,$n+1,et2
Code calculant l'expression M3 →$n
J et3
et1 : Code calculant l'expression M2 →$n
J et3
et2 : Code calculant l'expression M1 →$n
et3: nop
```

Partie 02 :

```
FONC FIB(n)
{
 Si n<2
        retour n ;
Sinon
          Retour FIB(n-1)+ FIB(n-2) ; (01 pts)
}
```

(01 pts)




```
FIB:  lw $8,n
li $9,2
blt $8,$9, et1
sw $31, -4 ($sp)
lw $8, n
li $9, 1
SUB $8,$8 ,$9
sw $8, -8 ($sp)
addi $sp, $sp, -8
jal FIB
sw $31, -4 ($sp)
```

```
lw $8, n
li $9, 2
SUB $8,$8 ,$9
sw $8, -8 ($sp)
addi $sp, $sp, -8
jal FIB
move $8,$9
sw $8, 8($sp)
sw $31, -4 ($sp)
addi $sp, $sp, 8
add $8,$8,$9
```

```
jr 31
j et2
et1:  lw $8, n
jr 31
et2 :nop ;
```

Université Mohamed Khider – Biskra.
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'Informatique

Module: **Compilation**
Niveau : **3ème Licence**
Année universitaire: **2014/2015**
Documents non autorisés

Rattrapage

Date : 24/02/2014 **Durée : 1h30m**

Exercice 01 (04 pts) :

On souhaite écrire un analyseur lexical pour le langage de commandes suivant :
– une commande est composée d'un nom de commande, suivi d'une liste optionnelle d'arguments, suivie d'une liste facultative d'options ;
– une liste d'arguments est une suite d'arguments ;
– une liste d'options est une suite non vide d'options encadrée par [et], à l'intérieur de laquelle les options sont séparées par virgule « , » ;
– une option est un caractère précédé d'un tiret ;
– un argument est un identificateur, de même qu'un nom de commande.
Par exemple :
macom arg1 arg2 [-a,-b] est une commande,
ainsi que macom [-f] et macom.

1. Dessinez un automate acceptant les chaines appartenant à ce langage.
2. Décrire ce langage A l'aide d'un fichier Lex.

Exercice 02 (08 pts) :

Dans les formules mathématiques du langage LATEX, une partie en indice est indiquée par le caractère souligné '_ '. Par exemple x_i représente x_i. Si l'indice n'est pas réduit à un seul caractère, il faut le mettre entre accolades. Par exemple x_{i+1} représente x_{i+1}. Les indices peuvent être imbriqués : x_{y_ i} représente x_{y_i} .
On considère la grammaire G suivante qui engendre les formules du langage LATEX:
FormuleLatex → ε | FormuleLatex Primaire | Primaire
Primaire → id | '_'
Indice Indice → '{' FormuleLatex '}' | id
avec VT = { id, ' ', '{', '}' }, où id représente un caractère quelconque autre que ' ', '{' et '}'.

1. Cette grammaire n'est pas LL(1) : pourquoi ?
2. Donner une grammaire G' équivalente à G et qui vous semble LL(1).
3. Donner la table d'analyse LL(1) de G'. Justifier en utilisant cette table que G' est une grammaire LL(1).
4. Construire l'automate LR pour G.
5. La grammaire G est-elle LR(0) ? Justifier.
6. La grammaire G est-elle SLR(1) ? Justifier.

Exercice 02 (08 pts) :

On souhaite introduire dans les expressions arithmétiques de MIL-0 deux fonctions standard à deux arguments entiers, MIN(a, b) et MAX(a, b) , dont les valeurs sont le minimum (resp. le maximum) de leurs deux arguments.
Exemple : une nouvelle version du calcul du PGCD par différences

```
VAR a ; b ; r.
LIRE a ; LIRE b {supposés tous deux positifs};
SI a = b ALORS ECRIRE a
SINON
TANTQUE MIN(a,b) <> MAX(a-b, b-a) FAIRE
r := MAX(a-b , b-a) ;
SI a<b ALORS b := r SINON a := r FINSI
FINTQ ;
ECRIRE MIN(a, b)
FINSI.
```

On les modélisera comme deux nouvelles formes d'expressions arithmétiques, au même titre que les constantes, les variables et les expressions binaires.

1. Etendre le lexeur et le parseur du cours. Les noms " MIN " et " MAX " seront traités comme des mots-clés. Attention ! Une forme comme : " a * MAX(x, y)+b " s'analyser comme :"((a * MAX(x, y))+b) ".
2. Etendre le générateur de code du cours.
3. Sur la base de votre proposition, traduire en assembleur MIPS le programme donné en exemple ci-dessus.
4. Construire l'arbre abstrait correspondant.

Université Mohamed Khider – Biskra.
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'Informatique

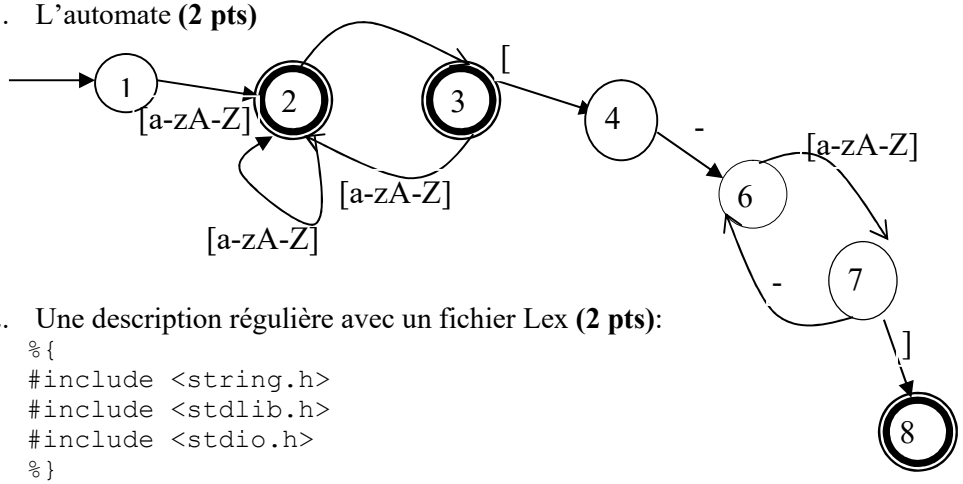
Module: **Compilation**
Niveau : **3ème Licence**
Année universitaire: **2014/2015**
Documents non autorisés

Corrigé type du Rattrapage

Fait le : 24/02/2014 **Durée : 1h30m**

Exercice 01 (04 pts) :

1. L'automate **(2 pts)**



2. Une description régulière avec un fichier Lex **(2 pts)**:

```
%{
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
%}
Lettre [a-zA-Z]
Id {Lettre}{Lettre}*
%%
 id(\  id)*(\[\-{lettre}(« , »\-{lettre})*\] ) ? {return (commentaire) ;}
%%
```

Exercice 02 (08 pts) :

On considère la grammaire G suivante qui engendre les formules du langage LATEX:
FL→ ε | FL P | P
P → id | '_' I
I → '{' FL '}' | id
avec VT = { id, '_ ', '{', '}' }, où id représente un caractère quelconque autre que ' ', '{' et '}'.

1. Il y'a la récursivité à gauche dans la règle : FL→ ε | FL P | P **(0.5 pts)**
2. Une grammaire G' équivalente à G **(1 pts)**

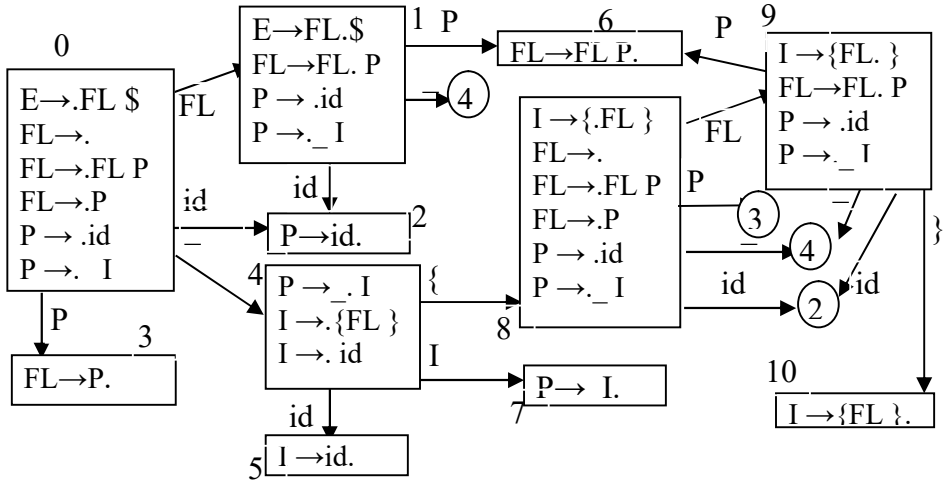
FL→ ε | P FL
P → id | '_' I
I → '{' FL '}' | id

1. Donner la table d'analyse LL(1) de G'. Justifier en utilisant cette table que G' est une grammaire LL(1) **(1.5 pts)**.

	Premiers	Suivants
FL	ε, id,_	$,}
P	id,_	id,_ , $,}
I	{, id	id,_ , $,}

	id	_	{	}	$
FL	FL→ P FL	FL→ P FL		FL→ ε	FL→ ε
P	P → id	P → _ I			
I	I → id		I → { FL }		

2. Construire l'automate LR pour G. **(2 pts)**



3. La grammaire G est-elle LR(0) ? Justifier.

	Premiers	Suivants
FL	ε, id,_	$,}
P	id,_	id,_ , $,}
I	{, id	id,_ , $,}

SLR(1)	id	_	{	}	$	**(01 pts)**	FL	P	I
0	d2	d4		r1	r1		1	3	
1		d4			acc			6	
2	r4	r4		r4	r4				
3				r3	r3				
4	d5		d8						7
5	r7	r7		r7	r7				
6				r2	r2				
7	r5	r5		r5	r5				
8	d2	d4		r1	r1		9	3	
9	d2	d4		d10				6	
10	r6	r6		r6	r6				

4. La grammaire G est-elle SLR(1) ? Justifier. **(01pts)**

LR(0)	id	_	{	}	$	**(01 pts)**	FL	P	I
0	d2/r1	d4/r1	r1	r1	r1		1	3	
1	d4/Accepter							6	
2	r4	r4	r4	r4	r4				
3	r3	r3	r3	r3	r3				
4	d5		d8						7
5	r7	r7	r7	r7	r7				
6	r2	r2	r2	r2	r2				
7	r5	r5	r5	r5	r5				
8	d2/r1	d4/r1	r1	r1	r1		9	3	
9	d2	d4		d10				6	
10	r6	r6	r6	r6	r6				

Exercice 02 (08 pts) :

On souhaite introduire dans les expressions arithmétiques de MIL-0 deux fonctions standard à deux arguments entiers, MIN(a, b) et MAX(a, b) , dont les valeurs sont le minimum (resp. le maximum) de leurs deux arguments. **Exemple** : une nouvelle version du calcul du PGCD par différences

```
VAR a ; b ; r.
LIRE a ; LIRE b {supposés tous deux positifs};
SI a = b ALORS ECRIRE a
SINON
TANTQUE MIN(a,b) <> MAX(a-b, b-a) FAIRE
r := MAX(a-b , b-a) ;
SI a<b ALORS b := r SINON a := r FINSI
FINTQ ;
ECRIRE MIN(a, b)
FINSI.
```

On les modélisera comme deux nouvelles formes d'expressions arithmétiques, au même titre que les constantes, les variables et les expressions binaires.

1. Etendre le lexeur et le parseur du cours. Les noms " MIN " et " MAX " seront traités comme des mots-clés. Attention ! Une forme comme : " a * MAX(x, y)+b " s'analyser comme :"((a * MAX(x, y))+b) ". **(2 pts)**
   Lexeur : deux lexèmes de plus correspondant aux deux mots-clés spécifiés par l'énoncé :

```
%%
…
MIN return (LMin);
MAX return (LMax);
...
%%
```

   Parseur : déclaration des deux Tokens :

```
%token LMin LMax
FACTEUR : '(' EXPAR ')' {$$ = $2;}
                | Cte {$$ = cons_ar_cte($1);}
                | VAR {$$ = cons_ar_var($1);}
                | LMin '(' EXPAR ',' EXPAR ')' {$$ = cons_min ($3, $5);}
                | LMax '(' EXPAR ',' EXPAR ')' {$$ = cons_max ($3, $5);}
;
```

2. Etendre le générateur de code du cours **(2 pts)**.

```
case min : {
Code Calculant arg1(e)→ k
Code Calculant arg1(e)→ k+1
blt $8, $9, etiqv
Move $8,k+1
j etiqfin
etiqv: Move $8 , k
etiqfin: nop
break;}
case max : {
Code Calculant arg1(e)→ k
Code Calculant arg1(e)→ k+1
bgt $8, $9, etiqv
Move $8, k+1
```

```
j etiqfin
etiqv: Move $8, k
etiqfin: nop
break;
}
```

3. Sur la base de votre proposition, traduire en assembleur MIPS le programme donné en exemple ci-dessus **(2 pts)**.

Hypothèses sur les adresses des variables : a = 0($30), b = 4($30), r = 8($30)

```
j e101 #boucle
e106: lw $8, 0($30) #a
lw $9, 4($30) #b
sub $8, $8, $9 # $8 = a-b
lw $9, 4($30) #b
lw $10, 0($30) #a
sub $9, $9, $10 # $9 = b-a
blt $8, $9, e102
j e103 # sinon, $8 contient le max
e102: move $8, $9 # si oui, on transfère le max
e103: nop # $8 contient le max(a-b, b-a)
sw $8, 8($30) #r
lw $8, 0($30) #a
lw $9, 4($30) #b
blt $8, $9, e104
lw $8, 8($30) #r
sw $8, 4($30) #b
j e105
e104: lw $8, 8($30) #r
sw $8, 0($30) #a
e105: nop # fin de conditionnelle
e101: lw $8, 0($30) #a
lw $9, 4($30) #b
bgt $8, $9, e107 # test inverse du précédent, pour Min
j e108
e107: move $8, $9
e108: nop # $8 contient le min(a, b)
lw $9, 0($30) #a # même séquence de code que ci-dessus
lw $10, 4($30) #b # un registre plus haut
sub $9, $9, $10
lw $10, 4($30) #b
lw $11, 0($30) #a
sub $10, $10, $11
blt $9, $10, e109
j e110
e109: move $8, $9
e110: nop # $9 contient le max(a-b, b-a)
bne $8, $9, e106 # test de la boucle
```

Construire l'arbre abstrait correspondant **(2 pts)**.