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: **2015/2016**
Documents non autorisés

Contrôle

Date : 13/01/2016 **Durée : 1h30m**

Exercice 01 (04 pts)

1. En Caml, les identificateurs sont des suites de lettres, chiffres, _ et ' (le caractère apostrophe), commençant par une lettre. Les identificateurs ne doivent pas contenir deux caractères soulignés successifs (_ _). Donnez une expression régulière pour reconnaître les identificateurs de Caml. Exemples : x, x', Le_Compteur, k0, K''1, mais pas ab_ _cd.

2. Le langage Ada permet d'écrire les nombres entiers comme par exemple 1_234_567, c'est à dire qu'on peut découper le nombre par tranche de 3 chiffres. Donner une expression régulière qui reconnaît ces nombres.

Exercice 02 (08 pts)

Soit G la grammaire suivante :

$A \rightarrow Bac \mid Bb$

$B \rightarrow Ad \mid Bc \mid \varepsilon$

1. Il est évident que cette grammaire n'est pas LL(1). Transformez-la en Grammaire G1de type LL(1).
2. Construire la table d'analyse LL(1) pour G1.
3. Analyser le mot « **cccdbac**» par un analyseur LL(1).;
4. Construire l'automate LR pour G.
5. G est-elle SLR(1)? Justifier rigoureusement en énumérant tous les conflits et en précisant leur type (shift/reduce ou reduce/reduce).

Exercice 03 (08 pts)

On veut ajouter la structure de donnée matrices ; MAT[i,j] pour être utilisée dans les ***expressions arithmétiques*** **:**

1. Quelles sont les modifications qui doivent être apportées aux différents fichiers du compilateur **(Lex, yacc, type.h).**
2. Proposer un modèle de code (d'instructions) MIPS permettant l'accès aux éléments de la matrice.

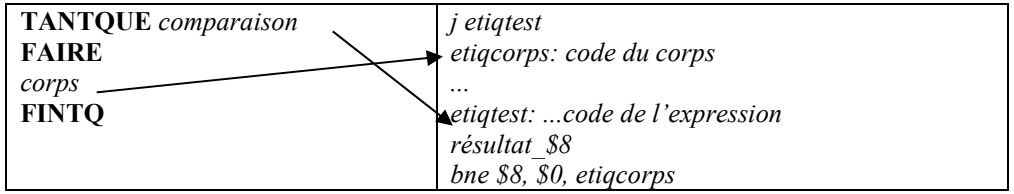
On veut utiliser cette structure (matrice) dans les instructions ***lecture*** et ***affectation***

3. Proposer la structure d'une déclaration de l'arbre syntaxique des instructions (INST_ARBRE) acceptant la matrice.
4. Ecrire en MNL un programme qui permet de trouver le max d'une matrice.
5. Suite à votre proposition construire l'arbre et donner le code MIPS complet correspondant (registre de départ c'est $8).

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



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 {$$ = CreerBin($2, $1, $3);}
  | G {};
G : TOKEN_INC_DEC V {$$ = CreerUn($1, $2);}
  |
  | F {};
V : '(' TOKEN_VARIABLE ')' {$$ = CreerVariable($2);}
  | TOKEN_VARIABLE {$$ = CreerVariable($1);};
F : '(' C ')' {$$ = $2;}
  | TOKEN_VARIABLE {$$ = CreerVariable($1);}
  | TOKEN_ENTIER {$$ = CreerConstante($1);};
%%
```

Type.h

```
enum EXPR_TYPE { CONSTANTE, VARIABLE, BINAIRE, UNAIRE,
TERNAIRE};
typedef struct _EXPR_ARBRE        ...
{                                 } forme;
enum EXPR_TYPE type;              } *INST_ARBRE;
union {
…
struct {
char op;
struct _EXPR_ARBRE *opg;
struct _EXPR_ARBRE *opd;
} bin;
…
} forme;
} *EXPR_ARBRE;


typedef struct _INST_ARBRE
{
enum INST_TYPE type;
union {
...
struct {
char *vecteur;
struct _EXPR_ARBRE *indice;
} lecture_vect;
```

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: **2015/2016**
Documents non autorisés

Corrigé type

Fait le : 13/01/2016 **Durée : 1h30m**

Exercice 01 (04 pts)

1. [A-Za-z]("_"?[A-Za-z0-9'])*"_"?

2. [0-9]{1,3}(_([0-9]{3}))* ou [0-9]([0-9][0-9]?)?(_[0-9][0-9][0-9])*

Exercice 02 (08 pts)

Soit G la grammaire suivante :

A→Bac|Bb

B → Ad|Bc | ε

1. Il est évident que cette grammaire n'est pas LL(1) **(01.5 pts)**.

Il existe une factorisation dans :A→Bac|Bb

A→Bac|Bb➔ A→B(ac|b)➔

A→BA'

A'→ac|b

Il existe une récursivité à gauche immédiate et indirecte dans :

B → Ad|Bc | ε

B → Ad|Bc | ε ➔ B →(Ad|ε)B'

B' →cB'| ε

A→BA'➔ A →(AdB'|B')A'➔ A →AdB'A'|B'A'➔A→ B'A'A''

A''→ dB'A'| ε

A→ B'A'A''

A'→ac|b

A''→ dB'A'| ε

B' →cB'| ε

Construire la table d'analyse LL(1) pour G1 **(01.5 pts).**

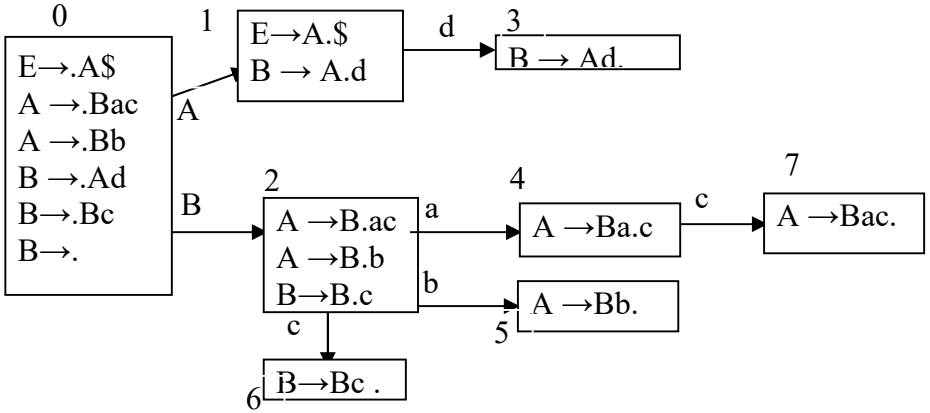
	Premiers	suivants
A	a,b,c	$
A'	a,b	d,$
A''	d, ε	$
B'	c, ε	a,b

	a	b	c	d	$
A	B'A'A''	B'A'A''	B'A'A''		
A'	ac	b			
A''				dB'A'	ε
B'	ε	ε	cB'		

2. Analyser le mot « **cccdbac**» par un analyseur LL(1) **(01pts)**;

Pile	**Chaine**	**Action**
$A	cccdbac$	dépiler, empiler A''A'B'
$ A''A'B'	cccdbac$	dépiler, empiler B'c
$ A''A'B'c	cccdbac$	dépiler, Avancer
$ A''A'B'	ccdbac$	dépiler, empiler B'c
$ A''A'B'c	ccdbac$	dépiler, Avancer
$ A''A'B'	cdbac$	dépiler, empiler B'c
$ A''A'B'c	cdbac$	dépiler, Avancer
$ A''A'B'	dbac$	erreur

3. Construire l'automate LR pour G **(02 pts)**.



4. G est-elle SLR(1) **(02 pts)**?

	Premiers	Suivants
A	a,b,c	$,d
B	a,b,c, ε	a,b,c

SLR(1)	a	b	c	d	$		A	B
0	R5	R5	R5				1	2
1				d3	acc			
2	d4	d5	d6					
3	R3	R3	R3					
4			d7					
5				d2	d2			
6	R4	R4	R4					
7				d1	d1			

Exercice 0 3 (08 pts)

On veut ajouter la structure de donnée matrices ; MAT[i,j] pour être utilisée dans les ***expressions arithmétiques*** **:**

1. Les modifications qui doivent être apportées aux différents fichiers du compilateur (Lex, yacc, type.h).

Lex

```
…
[?:= \[ ,\]] {return(yytext[0]);}
…
```

yacc

```
V : '(' V ')' { $$ = $2; }
| …
| TOKEN_VARIABLE '[' C',’C ']'{$$=CreerMatrice($1,$3,$5)};
F : '(' C ')' { $$ = $2; }
 |…
 |TOKEN_VARIABLE '[' C',’C ']'{$$=CreerMatrice($1,$3,$5)};
```

Type.h

```
enum EXPR_TYPE { … , MATRICE};
 typedef struct _EXPR_ARBRE {
 enum EXPR_TYPE type;
 union {
 …
 struct {
 char *NOMAT;
 struct _EXPR_ARBRE *INDL;
 struct _EXPR_ARBRE *INDC;
} MATRICE;
…
}
```

2. Proposer un modèle de code (d'instructions) MIPS permettant l'accès aux éléments de la matrice.

```
Code pour calculer son indice de la ligne (qui est une
expression) résultat $n
Code pour calculer son indice de la ligne (qui est une
expression) résultat $n+1
mul $n, $n, (nombre de colonnes)
add $n, $n, $n+1
li $n+1, 4
mul $n, $n, $n+1
add $n, $n, $30
lw $n, d($n)
```

On veut utiliser cette structure (matrice) dans les instructions ***lecture*** et ***affectation***

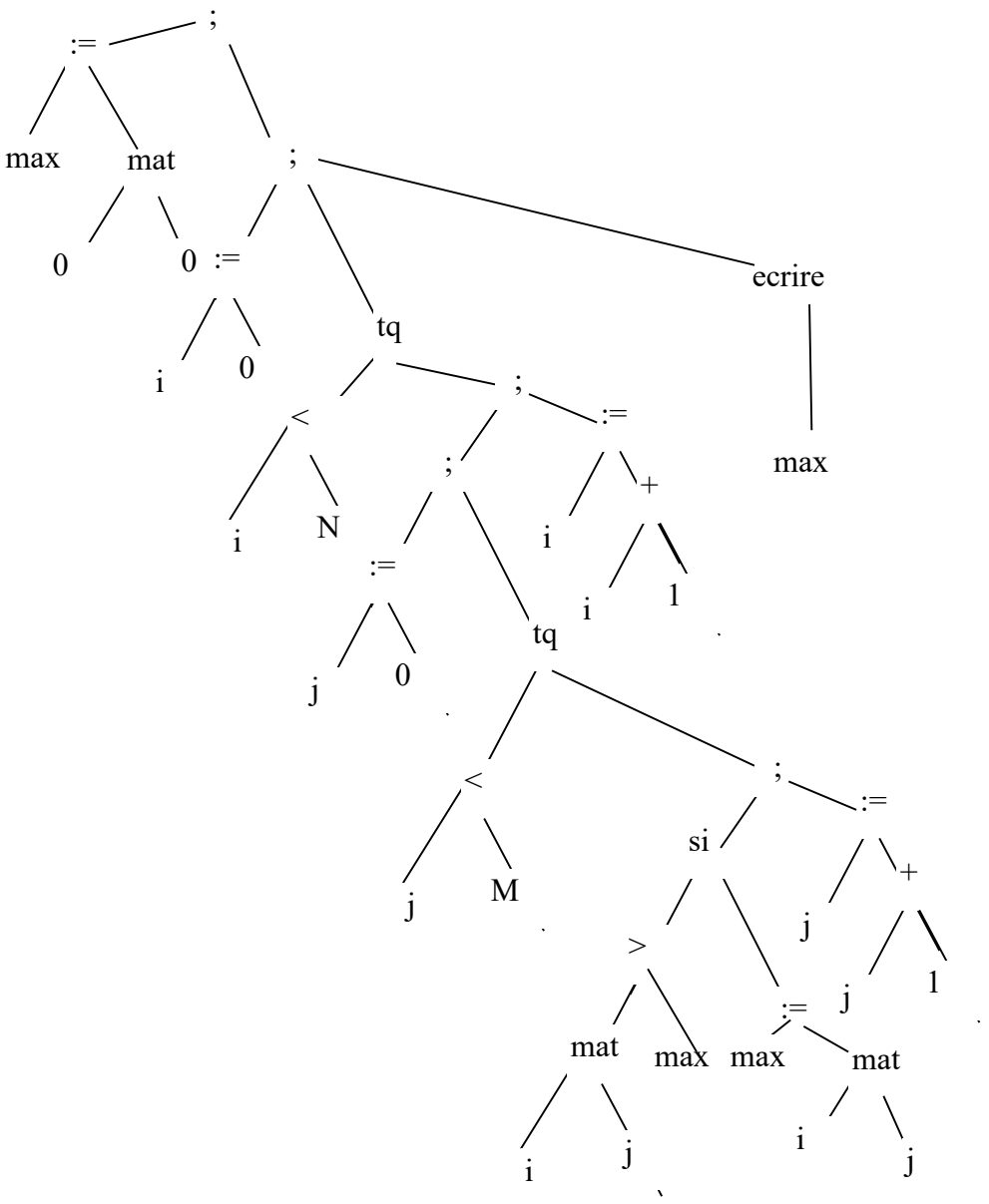
3. Proposer la structure d'une déclaration de l'arbre syntaxique des instructions (INST_ARBRE) acceptant la matrice.

```
typedef struct _INST_ARBRE {
enum INST_TYPE type;
union {
...
struct {
      char *Matrice;
      struct _EXPR_ARBRE *indiceL;
      struct _EXPR_ARBRE *indiceC;
      } lecture_Mat;

struct {
       char *Matrice;
       struct _EXPR_ARBRE *indiceL;
       struct _EXPR_ARBRE *indiceC;
       struct _EXPR_ARBRE *expr
      } Affect_Mat;

...
} forme;
} *INST_ARBRE;
```

4. Un programme MNL permettant de trouver le max d'une matrice.

```
Const N=5,M=5 ;
Var i,j,Max
Matrice Mat[N,M]
Max :=MAT[0,0] ;
i :=0 ;
Tantque i<N faire
{   j :=0 ;
 Tantque j<M faire
  {   Lire MAT[i,j];
       j :=j+1 ;
    }
   i:=i+1 ;
}
i :=0 ;
Tantque i<N faire
{   j :=0
 Tantque j<M faire
  { Si(MAT[i,j]>max)
        max= MAT[i,j];
    j :=j+1 ;
    }
   i:=i+1 ;
}
Ecrire Max ;
```

5. L'arbre syntaxique



6. Le code MIPS complet correspondant (registre de départ c'est $8).

```
.data
MEM: .space 48
.text
main: la $30, MEM
… #lecture de la
matrice
Li $8,0($30)
li $9, 4($30)
li $10,3
mul $8, $8,$10
add $8, $8, $9
li $9, 4
mul $8, $8, $9
add $8, $8, $30
lw $8, 12($8)
sw $8,12($30)
li $8,0
sw $8, 0($30)
j et1
et2:li $8,0
sw $8, 0($30)
j et3
et4 :lw $8,0($30)
lw $9, 4($30)
li $10,3
mul $8, $8,$10
add $8, $8, $9
li $9, 4
mul $8, $8, $9
add $8, $8, $30
lw $8, 12($8)
lw $8, 8($30)
bgt $8, $9, et5
j et6
et5: lw $8,0($30)
lw $9, 4($30)
li $10,3
mul $8, $8,$10
add $8, $8, $9
li $9, 4
mul $8, $8, $9
add $8, $8, $30
lw $8, 12($8)
sw $8, 8($30)
et6 :nop
lw $8, 4($30)
li $9,1
ADD $8,$8,$9
sw $8, 4($30)
et3:lw $8, 4($30)
li $9, 3
Blt $8, $9, et4
lw $8, 0($30)
li $9,1
ADD $8,$8,$9
sw $8, 0($30)
Et1: lw $8,0($30)
li $9,3
Blt $8, $9,et2
 lw $8, 12($8)
move $8,$4
li $2,1
syscall
li $2,10
syscall
```

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: **2015/2016**
Documents non autorisés

Rattrapage

Date : 28/02/2016 **Durée : 1h30m**

Exercice 01 (04 pts) :

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

S → AB

A → BA|ba|C

B → ac|a

C → cC|aC

1. Qu'est ce que empêche cette grammaire d'être LL(1). Transformez-la en Grammaire G1de type LL(1).
2. Construire la table d'analyse LL(1) pour G1.
3. Construire l'automate LR pour G.
4. G est-elle SLR(1)? Justifier rigoureusement en énumérant tous les conflits et en précisant leur type (shift/reduce ou reduce/reduce).
5. Analyser le mot « **cccdbac**» par un analyseur LL(1).;

Exercice 02 (08 pts) :

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: **2015/2016**
Documents non autorisés

Corrigé type du Rattrapage

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

Exercice 01 (04 pts) :