Université Mohamed Khider – Biskra. Module: **Compilation**
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie Niveau : **3ème Licence**
Département d'Informatique Année universitaire: **2012/2013**
**Documents non autorisés**

# Contrôle

**Date : 23/01/2013** **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 pair de chiffres pairs,
   - ou composés de nombre impair de chiffres impairs.

**Exemples d'entiers acceptés:** 224466 246820 135 9975311
**Exemples d'entiers non acceptés:** 024666 2410 1357 13257

2. Donner une expression rationnelle étendue permettant de reconnaître des durées (strictement inférieures à 24h) en heures, minutes et secondes respectant les contraintes suivantes :
   - Une durée s'exprime avec le format `--h--m--s`.
   - Les secondes et minutes sont comprises entre 0 et 59 et composées d'un ou deux chiffres.
   - Les heures sont comprises entre 0 et 23 et composées d'un ou deux chif
   - Les heures ne peuvent être présentes que si les minutes y sont et les minutes ne peuvent être présentes que si les secondes y sont.

**Exemples de durées acceptées :** 05s 05m00s 02h3m04s 2h03m4s
**Exemples de durées non acceptées:** 05m : sans la présence des secondes

**Exercice 02 (08 pts)**

On considère la grammaire G de terminaux {a; x; d; e}, de non-terminaux {S; A;B}, d'axiome S, et de productions :
S→eAd | eB
A→ aA | a | x
B → x

1. Il est évident que cette grammaire n'est pas LL(1) : pourquoi (en détail)? Transformez-la en Grammaire G1de type LL(1).
2. Construire la table d'analyse LL(1) pour G1.
3. Analyser le mot « *eaxd* » par un analyseur SLR(1).;
4. Construire l'automate LR pour G
5. G est-elle LR(0) ? SLR(1) ? Justifier rigoureusement en énumérant tous les conflits et en précisant leur type (shift/reduce ou reduce/reduce).

**Exercice 03 (08 pts)**

Dans certain langage, Carre(A) est introduit comme une fonction mathématique permettant de rendre la valeur **A à la puissance de 2**
Par exemple : carre(3 + 2 *1) = carre(5)=25.
On veut introduire ce **carre(exp)** dans le MiniLangage

1. Modifier ces fichiers d'entrée, ( Lex et Yacc) pour accepter cette expression.
2. Proposer une déclaration de la structure de cette expression dans l'arbre syntaxique.
3. Proposer une règle de génération du code MIPS de cette expession.
4. En utilisant ce nouvel opérateur (carre), écrire dans minilangage un programme permettant de calculer la valeur $x^{2*n}$ pour n'importe quelle valeur de n>=0.
5. Sur la base de votre proposition construire l'arbre syntaxique et donner le code MIPS correspond au programme de la question précédente.

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

| Nom | N° | 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,<br>retourne l'adresse du début dans v0. |
| exit | 10 | arrêt du programme en cours d'exécution |

| 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 |

**Fichier yacc**

```
……
%%
EE : C {a = $1;};
C : E '?' C ':' C{$$ = CreerTer($1, $3, $5);}
| E {};
E : E TOKEN_PLUS_MOINS T {$$ = CreerBin($2, $1, $3);}
  | T {/* par defaut, $$ = $1 */};
T : T TOKEN_MULT_DIV G {$$ = CreerBin($2, $1, $3);}
| G {};
G : TOKEN_INC_DEC TOKEN_VARIABLE {$$ = CreerUn($1, $2);}
| F {} ;
F : '(' E ')' {$$ = $2;}
| TOKEN_VARIABLE {$$ = CreerVariable($1);}
| TOKEN_ENTIER {$$ = CreerConstante($1);};
%%
……
```

Université Mohamed Khider – Biskra. Module: **Compilation**
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie Niveau : **3ème Licence**
Année universitaire: **2012/2013**
Département d'Informatique **Documents non autorisés**

# Corrigé type

**Fait le : 23/01/2013** **Durée : 1h30m**

## Exercice 01 (04 pts)

1. Donner une expression régulière étendue permettant de reconnaître des entiers naturels **(02 pts)**:
   ((2|4|6|8) (0|2|4|6|8)( (0|2|4|6|8){2})*)|((1|3|5|7|9)( (1|3|5|7|9){2})*)
2. Donner une expression rationnelle étendue permettant de reconnaître des durées (strictement inférieures à 24h) en heures, minutes et secondes respectant les contraintes suivantes **(02 pts)** :
   (([0-1][0-9]|2[0-3]h)? [0-5][0-9]m) ?[0-5][0-9]s

## Exercice 02 (08 pts)

On considère G :

S→eAd | eB
A→ aA | a | x
B → x

1. Cette grammaire n'est pas LL(1) parceque il y a deux cas de factorisation **(01 pts)**:
   S→eAd | eB
   A→ aA | a
   Et un cas de conflit (apparaisse après la création de la table)
2. Grammaire G1 de type LL(1) **(01 pts)**.
   1. S→eS'
   2. S'→ aA'd| xS''
   3. S''→d | ε
   4. A→ aA'| x
   5. A'→A|ε
3. La table d'analyse LL(1) pour G1 **(01.5 pts)**.

| | Premiers | Suivant |
|---|---|---|
| S | e | $ |
| S' | a,x | $ |
| S'' | d, ε | $ |
| A | a,x | d |
| A' | a,x, ε | d |
| | | |

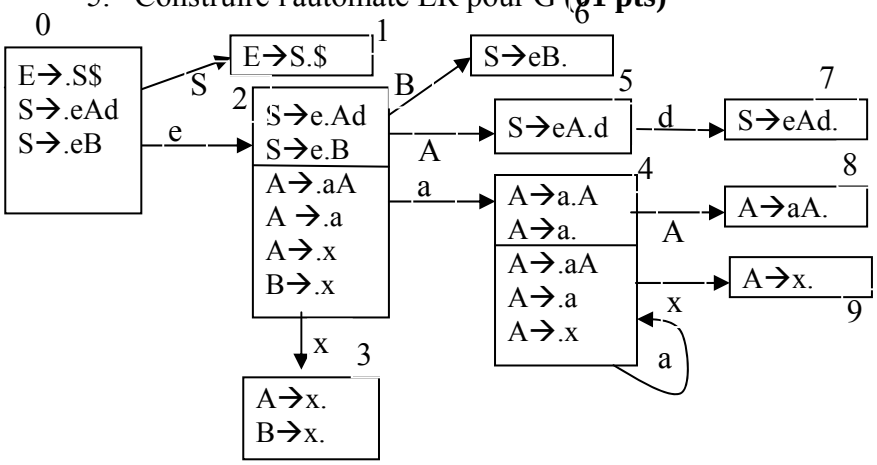
| | a | x | d | e | $ |
|---|---|---|---|---|---|
| S | | | | eS' | |
| S' | aA'd | xS'' | | | |
| S'' | | | d | | ε |
| A | aA' | x | | | |
| A' | A | A | ε | | |



4. Analyser le mot « eaxd » par un analyseur LL(1) **(01 pts)**.

| Pile | Chaine | Action |
|---|---|---|
| $S | *eaxd*$ | S →eS' |
| $S'e | *eaxd*$ | dépiler, avancer |
| $S' | *axd*$ | S'→aA'd |
| $dA'a | *axd*$ | dépiler, avancer |
| $dA' | *xd*$ | A'→A |
| $dA | *xd*$ | A→x |
| $dx | *xd*$ | dépiler, avancer |
| $d | *d*$ | dépiler, avancer |
| $ | $ | Accépter |

5. Construire l'automate LR pour G (**01 pts**)




1. S→eAd
2. S→ eB
3. A→ aA
4. A→ a
5. A→ x
6. B → x

| | Premiers | Suivant |
|---|---|---|
| S | e | $ |
| A | a,x | d |
| B | x | $ |

6. G n'est LR(0) parcequ'il y a un conflit de type réduire/réduire dans l'état 3 (r5/r6) et un conflit de type décaler/réduire (d4/r4) et (d9/r4) (voir la table)
7. G est SLR(1) parcequ'il n'y a aucun cas de conflits (voir la table) **(0.5 pts)**.

| LR(0) | a | x | d | e | $ | **(01 pts)** | S | A | B |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | d2 | | | 1 | | |
| 1 | Accépter | | | | | | | 5 | 6 |
| 2 | d4 | d3 | | | | | | | |
| 3 | réduire 5/6 | | | | | | | | |
| 4 | d4 | d9 | réduire 4 | | | | | 8 | |
| 5 | | | d7 | | | | | | |
| 6 | réduire 2 | | | | | | | | |
| 7 | réduire 1 | | | | | | | | |
| 8 | réduire 3 | | | | | | | | |
| 9 | réduire 5 | | | | | | | | |

| SLR(1) | a | x | d | e | $ | **(01 pts)** | S | A | B |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | d2 | | | 1 | | |
| 1 | | | | | ACC | | | 5 | 6 |
| 2 | d4 | d3 | | | | | | | |
| 3 | | | r5 | | r6 | | | | |
| 4 | d4 | d9 | r4 | | | | | 8 | |
| 5 | | | d7 | | | | | | |
| 6 | | | | | r2 | | | | |
| 7 | | | | | r1 | | | | |
| 8 | r3 | r3 | | | | | | | |
| 9 | | | r5 | | | | | | |

**Exercice 03 (08 pts)**

Dans certain langage, Carre(A) est introduit comme une fonction mathématique permettant de rendre la valeur **A à la puissance de 2**
On veut introduire ce **carre(exp)** dans le MiniLangage

1. Modification des fichiers Lex et Yacc **(1.5 pts)**.

**Fichier Lex**

```
%%
.........
carre {yylval.caractere = yytext[0]; return(TOKEN_Carre);}
{ER_VARIABLE} {
yylval.chaine = strcpy((char *)malloc(yyleng+1), yytext);
return(TOKEN_VARIABLE);}
.........
```



**Fichier yacc**

```
%%
.....
F : '(' E ')' {$$ = $2;}
| TOKEN_VARIABLE {$$ = CreerVariable($1);}
| TOKEN_ENTIER {$$ = CreerConstante($1);};
| TOKEN_Carre(C);{$$ = CreerUn($1,$3) ;}
%%
.....
```

2. une déclaration de la structure de cette **(0.5 pts)**.
   Puisque l'opérateur carre est une opération unaire on propose la structure suivante :

```
typedef struct _EXPR_ARBRE {
  enum EXPR_TYPE type;
  union {
    int   val;
    char *nom;
    struct {
      char             op;
      struct _EXPR_ARBRE *u;
    } un;
   …..
  } forme;
} *EXPR_ARBRE;
```

3. Proposer une règle de génération du code MIPS de cette expession **(01 pts)**.

*Code pour calculer son paramètre (qui est une expression) résultat $8*
*Move $8,$9*
*Mult $8,$8,$9*

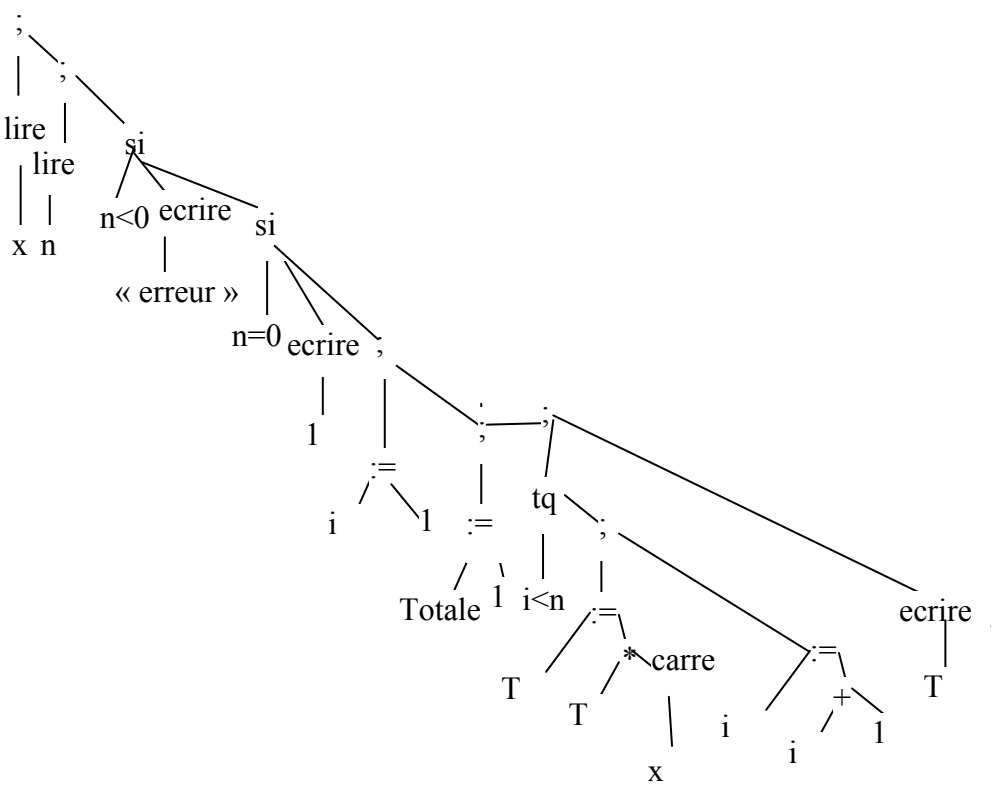
4. Un programme en minilangage permettant de calculer $x^{2*n}$ **(01 pts)**; :

```
var x,n,i,T

Ecrire"Donner la valeur de x, n:\n"
lire x ;
lire n ;
ecrire « entrez la valeur de x et n » ;
si n<0 alors
      ecrire « erreur » ;
sinon
      si n=0 alors ecrire 1 ;
            sinon
               i :=1 ;
               Totale :=1 ;
               tantque (i<n) faire
                       T=T*carre(x) ;
                       i :=i+1 ;
               Fin Tq ;
               ecrire T ;
```

```
            Fin si ;
        Finsi ;
    .
```

5. Construire l'arbre syntaxique **(02pts)**:




6. le code MIPS correspond au programme de la question 5 **(02 pts)**

```
.data
MEM: .space 16
CHAINE0: .asciiz "Donner la valeur de x, n:\n"
CHAINE1: .asciiz "Erreur \n"
      .text
main:  la $30, MEM
      la $4, CHAINE0
      li $2, 4
      syscall
      li $2, 5
      syscall
```



```
      sw $2, 0($30)
      li $2, 5
      syscall
      sw $2, 4($30)
      lw $8,4($30)
      blt $8,$0,et0
      lw $8,4($30)
      be $8,$0,et3
      li $8,1
      sw $8,8($30)
      li $8,1
      sw $8,12($30)
      j et5
et6 : lw $8,12($30)
      lw $9,0($30)
      move $9,$10
      mult $9,$9,$10
      mult $8,$8,$9
      sw $8,12($30)
et5 : lw $8,8($30)
      lw $9,4($30)
      blt $8,$9,et6
      lw $8,12($30)
      move $4, $8
      li $2, 4
      syscall
      j et4
et3 : li $8,1
      move $4, $8
      li $2, 4
      syscall
      j et4
et4 : nop
      j et1
et0 : la $4, CHAINE1
      li $2, 4
      syscall
et1 : nop
      li $2, 10
      syscall
```