Faculté des Sciences
Aix*Marseille Université

**Année universitaire 2017/2018**

Site : ☒ Luminy ☐ St-Charles ☐ St-Jérôme ☐ Cht-Gombert ☒ Aix-Montperrin ☐ Aubagne-SATIS
Sujet de : ☐ 1er semestre ☒ 2ème semestre ☐ Session 2 Durée de l'épreuve : 2h
Examen de : L3 Nom du diplôme : Licence d'Informatique
Code du module : ENSIN6U1 Libellé du module : Compilation
Calculatrices autorisées : NON Documents autorisés : NON

---

## Conventions

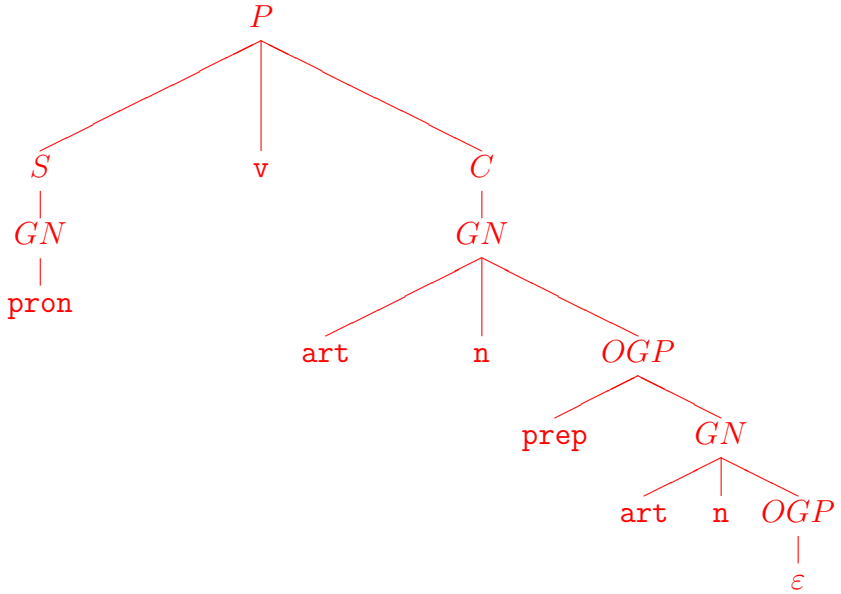
- Axiome : symbole non terminal de la partie gauche de la première production de la grammaire
- Symboles non terminaux : lettres *MAJUSCULES ITALIQUES*
- Symboles terminaux : lettres `minuscules true-type` ou caractères spéciaux simples

## 1 Analyse syntaxique (5pt)

La grammaire $G_{\text{phrase}}$ ci-dessous génère des phrases simples du français du type *sujet-verbe-complément*. Une phrase $P$ est composée d'un sujet $S$, d'un verbe `v` et d'un complément $C$. Le sujet $S$ est obligatoirement un groupe nominal $GN$ et le complément $C$ est un groupe nominal $GN$ optionnel. Un groupe nominal $GN$ est composé d'un article `art` suivi d'un nom `n` et d'un groupe prépositionnel optionnel $OGP$. Alternativement, un groupe nominal $GN$ est tout simplement un pronom `pron`. Le groupe prépositionnel optionnel $OGP$, quand il n'est pas vide, est composé d'une préposition `prep` suivie d'un groupe nominal $GN$.

1 $P \rightarrow S$ `v` $C$
2 $S \rightarrow GN$
3 $C \rightarrow GN$
4 $C \rightarrow \varepsilon$
5 $GN \rightarrow$ `art n` $OGP$
6 $GN \rightarrow$ `pron`
7 $OGP \rightarrow$ `prep` $GN$
8 $OGP \rightarrow \varepsilon$

**Q1.1** Dessiner l'arbre de dérivation du mot `pron v art n prep art n`. (1pt)



**Q1.2** Calculer les ensembles PREMIER et SUIVANT pour chaque non-terminal de la grammaire $G_{\text{phrase}}$. (1pt)

| Non-terminal | PREMIER | SUIVANT |
|---|---|---|
| $P$ | `art pron` | $\perp$ |
| $S$ | `art pron` | `v` |
| $C$ | `art pron` $\varepsilon$ | $\perp$ |
| $GN$ | `art pron` | $\perp$ `v` |
| $OGP$ | `prep` $\varepsilon$ | $\perp$ `v` |

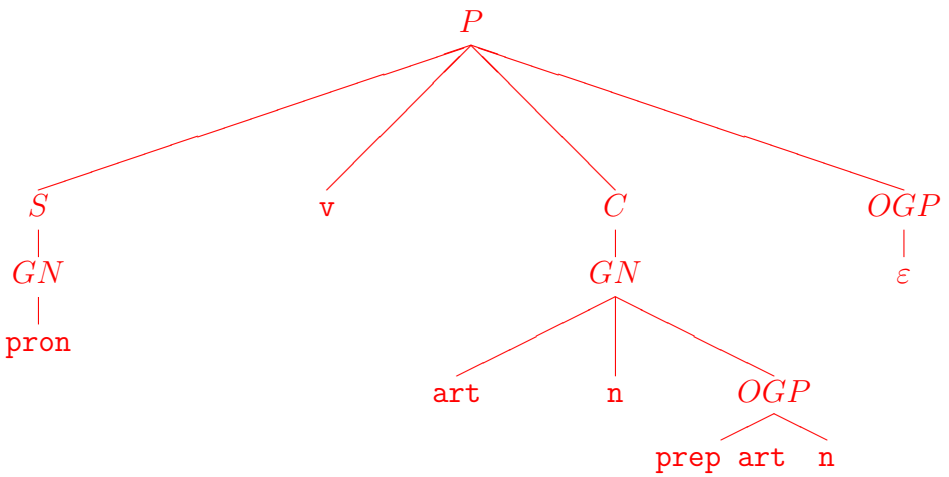
**Q1.3** Dessiner la table d'analyse $LL(1)$ de la grammaire $G_{\text{phrase}}$. (1pt)

| | `art` | `pron` | `v` | `n` | `prep` | $\perp$ |
|---|---|---|---|---|---|---|
| $P$ | 1 | 1 | | | | |
| $S$ | 2 | 2 | | | | |
| $C$ | 3 | 3 | | | | 4 |
| $GN$ | 5 | 6 | | | | |
| $OGP$ | | | 8 | | 7 | 8 |

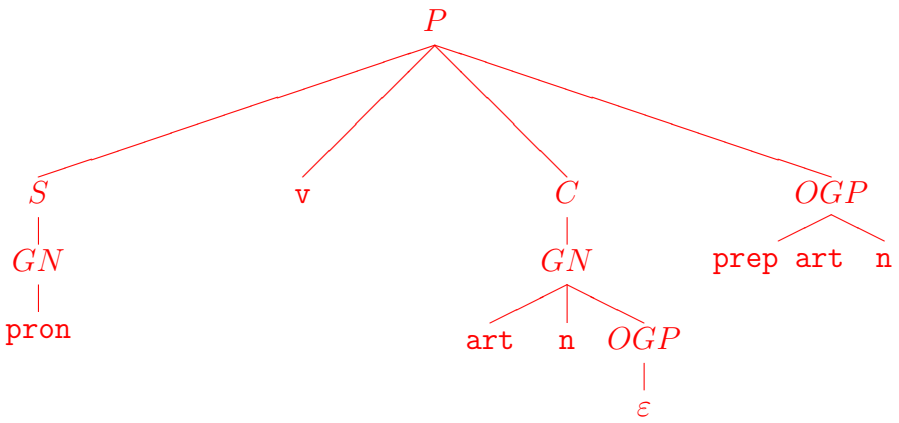
**Q1.4** Simuler les cinq premières étapes de l'analyse $LL(1)$ du mot `pron v art n prep art n`. Les configurations sont représentées par le triplet $(w, \alpha, y)$, où $w$ est la partie de la bande de lecture qui commence au caractère sous la tête de lecture (ce qui reste à analyser), $\alpha$ est la pile et $y$ est la séquence de symboles de sortie (numéros des productions appliquées). (1pt)

(`pron v art n prep art n` $\perp$ , $\varepsilon$ ) $\mapsto$
(`pron v art n prep art n` $\perp$, $P$ $\perp$ , $\varepsilon$ ) $\mapsto$
(`pron v art n prep art n` $\perp$, $S$ `v` $C$ $\perp$ , 1 ) $\mapsto$
(`pron v art n prep art n` $\perp$, $GN$ `v` $C$ $\perp$ , 1-2 ) $\mapsto$
(`pron v art n prep art n` $\perp$, `pron v` $C$ $\perp$ , 1-2-6 ) $\mapsto$
(`v art n prep art n` $\perp$, `v` $C$ $\perp$ , 1-2-6 ) $\mapsto$
(`art n prep art n` $\perp$, $C$ $\perp$ , 1-2-6 ) $\mapsto$
(`art n prep art n` $\perp$, $GN$ $\perp$ , 1-2-6-3 ) $\mapsto$
(`art n prep art n` $\perp$, `art n` $OGP$ $\perp$, 1-2-6-3-5 ) $\mapsto$
(`n prep art n` $\perp$, `n` $OGP$ $\perp$ , 1-2-6-3-5 ) $\mapsto$
(`prep art n` $\perp$, $OGP$ $\perp$ , 1-2-6-3-5 ) $\mapsto$
(`prep art n` $\perp$, `prep` $GN$ $\perp$ , 1-2-6-3-5-7 ) $\mapsto$
(`art n` $\perp$, $GN$ $\perp$ , 1-2-6-3-5-7 ) $\mapsto$
(`art n` $\perp$, `art n` $OGP$ $\perp$, 1-2-6-3-5-7-5 ) $\mapsto$
(`n` $\perp$, `n` $OGP$ $\perp$ , 1-2-6-3-5-7-5 ) $\mapsto$
($\perp$, $OGP$ $\perp$ , 1-2-6-3-5-7-5 ) $\mapsto$
($\perp$, $\perp$ , 1-2-6-3-5-7-5-8) $\mapsto$

**Q1.5** On modifie la première règle qui est maintenant $P \to S$ `v` $C$ $OGP$. Montrer que la nouvelle grammaire est ambiguë. (1pt)
Le mot `pron v art n prep art n` a désormais 2 arbres de dérivation possibles :


P
S
v
C
OGP
GN
GN
ε
pron
art
n
OGP
prep art n

et


P
S
v
C
OGP
GN
GN
prep art n
pron
art
n
OGP
ε

## 2 Grammaires attribuées : quadtrees (10pt)

Soit une image carrée de $N \times N$ pixels, où $N$ est une puissance de deux et dont le coin supérieur gauche a pour coordonnées $(0, 0)$. On peut la représenter par une structure d'arbre, appelée *quadtree*, dans laquelle chaque nœud a exactement zéro ou quatre fils. L'idée est la suivante : si une image associée à un nœud n'est pas homogène (d'une seule couleur), on la découpe en quatre carrés de même taille et les fils du nœud correspondant sont les *quadtrees* associés aux quatre carrés. Si au contraire l'image est homogène, alors elle est caractérisée par une information unique, sa couleur ; dans ce cas, le nœud porte cette information et n'a pas de fils.
Par exemple, si on parcourt les quatre quarts d'une image en décrivant un Z, et si on suppose que les dix régions en lesquelles on a découpé l'image de la figure 1 sont homogènes, de couleurs `r`, `g` et `b`, alors cette figure peut être représentée par le *quadtree* de la figure 2.

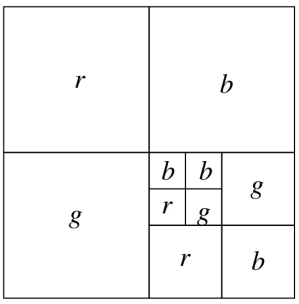



*Figure 1*

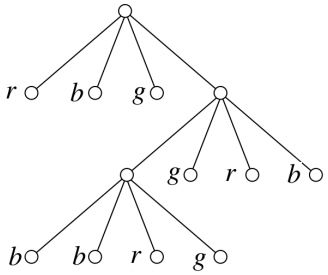



*Figure 2*

Un quadtree peut aussi être représenté sous la forme d'une expression parenthésée. Le quadtree de la figure 2, par exemple, est représenté de la façon suivante : `(r b g ((b b r g) g r b))`
Le langage des quadtrees peut être généré à l'aide de la grammaire $G_{\text{quad}}$, d'axiome $S$, définie ci-dessous :

$S \rightarrow Q$
$Q \rightarrow$ `(` $Q\ Q\ Q\ Q$ `)`
$Q \rightarrow$ `r`
$Q \rightarrow$ `b`
$Q \rightarrow$ `g`

Dans chacune des questions suivantes on cherche à répondre à une question concernant un quadtree généré par la grammaire. Pour chacune, il vous est demandé de définir un ou plusieurs attributs ainsi que les actions sémantiques associées aux règles de $G_{\text{quad}}$ permettant de calculer la valeur des attributs. Tous les attributs sont à valeurs entières. Pour chaque attribut défini, indiquer : ce qu'il représente et s'il est hérité ou synthétisé. Pour chaque question, vous pourrez utiliser les attributs définis dans les questions précédentes.

**Q2.1** Quelle est la longueur du côté de chaque carré $Q$ du quadtree ? (2pt)
On définit l'attribut $c$ hérité, dont les valeurs sont calculées de la manière suivante :

| | | | |
|---|---|---|---|
| $S$ | $\rightarrow$ | $Q$ | $Q.c = N$ |
| $Q$ | $\rightarrow$ | `(` $Q_1$ | $Q_1.c = Q.c/2$ |
| | | $Q_2$ | $Q_2.c = Q.c/2$ |
| | | $Q_3$ | $Q_3.c = Q.c/2$ |
| | | $Q_4$ `)` | $Q_4.c = Q.c/2$ |
| $Q$ | $\rightarrow$ | `r` \| `g` \| `b` | |

**Q2.2** Quelles sont les coordonnées du coin haut gauche de chaque carré $Q$ du quadtree ? (2pt)
On définit les deux attributs hérités $x$ et $y$ qui représentent l'abscisse et l'ordonnée du coin haut gauche de chaque carré :

| | | | | |
|---|---|---|---|---|
| $S$ | $\to$ | $Q$ | $Q.x = 0$ | $Q.y = 0$ |
| $Q$ | $\to$ | ( $Q_1$ | $Q_1.x = Q.x$ | $Q_1.y = Q.y$ |
| | | $Q_2$ | $Q_2.x = Q.x + Q_2.c$ | $Q_2.y = Q.y$ |
| | | $Q_3$ | $Q_3.x = Q.x$ | $Q_3.y = Q.y + Q_3.c$ |
| | | $Q_4$ ) | $Q_4.x = Q.x + Q_4.c$ | $Q_4.y = Q.y + Q_4.c$ |
| $Q$ | $\to$ | r \| g \| b | | |

**Q2.3** Combien de carrés homogènes (c'est-à-dire, de couleur uniforme `r`, `g` ou `b`) comporte le quadtree ? (2pt)

On définit l'attribut $x$ synthétisé, dont les valeurs sont calculées de la manière suivante :

| | | | |
|---|---|---|---|
| $S$ | $\to$ | $Q$ | $S.x = Q.x$ |
| $Q$ | $\to$ | ( $Q_1$ | $Q.x = Q_1.x + Q_2.x + Q_3.x + Q_4.x$ |
| | | $Q_2$ | |
| | | $Q_3$ | |
| | | $Q_4$ ) | |
| $Q$ | $\to$ | r | $Q.x = 1$ |
| $Q$ | $\to$ | b | $Q.x = 1$ |
| $Q$ | $\to$ | g | $Q.x = 1$ |

**Q2.4** Quelle est la longueur du côté du plus grand carré homogène du quadtree ? (Xpt)

On définit l'attribut $l$ synthétisé, dont les valeurs sont calculées de la manière suivante :

| | | | |
|---|---|---|---|
| $S$ | $\to$ | $Q$ | $S.l = Q.l$ |
| $Q$ | $\to$ | $(Q_1,$ | $Q.l = \max(Q_1.l, Q_2.l, Q_3.l, Q_4.l)$ |
| | | $Q_2,$ | |
| | | $Q_3,$ | |
| | | $Q_4)$ | |
| $Q$ | $\to$ | $r$ | $Q.l = Q.c$ |
| $Q$ | $\to$ | $b$ | $Q.l = Q.c$ |
| $Q$ | $\to$ | $g$ | $Q.l = Q.c$ |

**Q2.5** Combien de pixels de couleur `r` comporte le quadtree ? (2pt)

On définit l'attribut $r$ synthétisé, dont les valeurs sont calculées de la manière suivante :

| | | | |
|---|---|---|---|
| $S$ | $\to$ | $Q$ | $S.r = Q.r$ |
| $Q$ | $\to$ | ( $Q_1$ | $Q.r = Q_1.r + Q_2.r + Q_3.r + Q_4.r$ |
| | | $Q_2$ | |
| | | $Q_3$ | |
| | | $Q_4$ ) | |
| $Q$ | $\to$ | r | $Q.r = Q.c \times Q.c$ |
| $Q$ | $\to$ | b | $Q.r = 0$ |
| $Q$ | $\to$ | g | $Q.r = 0$ |

**Q2.6** Quelle est la couleur dominante d'un quadtree ? (s'il existe plusieurs couleurs correspondant au même nombre de pixels, il suffit d'en indiquer une) (2pt)

On définit l'attribut $d$ synthétisé, et on suppose l'existence des attributs $g$ et $b$. La valeur de $d$ est calculée de la manière suivante :

| | | | |
|---|---|---|---|
| $S$ | $\to$ | $Q$ | $S.d = \max(Q.r, Q.g, Q.b)$ |
| $Q$ | $\to$ | ( $Q_1$ | |
| | | $Q_2$ | |
| | | $Q_3$ | |
| | | $Q_4$ ) | |
| $Q$ | $\to$ | r \| g \| b | |

**Q2.7** Quelle est la profondeur d'un quadtree ? Le nœud racine d'un quadtree a pour profondeur 0, ses fils ont pour profondeur 1 . . . La profondeur d'un quadtree est la profondeur de son fils le plus profond. (2pt)

| | | | |
|---|---|---|---|
| $S$ | $\rightarrow$ | $Q$ | $S.p = Q.p$ |
| $Q$ | $\rightarrow$ | `(` $Q_1$ | $Q.p = \max(Q_1.p, Q_2.p, Q_3.p, Q_4.p) + 1$ |
| | | $Q_2$ | |
| | | $Q_3$ | |
| | | $Q_4$ `)` | |
| $Q$ | $\rightarrow$ | `r` \| `g` \| `b` | $Q.p = 0$ |

## 3 Génération de code assembleur (5pt)

Soit le programme $L$ suivant :

```
carre(entier $x) { retour $x * $x;}
main()entier $x; {$x = 2; carre($x);}
```

**Q3.1** Dessiner l'arbre abstrait correspondant au programme. (Note : ne représentez pas votre arbre sous la forme XML, cela prend trop de temps. Dessinez-le directement.)

Voici la liste de certains types de nœuds de l'arbre abstrait

| | |
|---|---|
| `appel` | appel de fonction |
| `foncDec` | déclaration de fonction |
| `instr_X` | instruction de type X |
| `XExp` | expression de type X |
| `l_dec` | liste de déclarations |
| `l_exp` | liste d'expressions |
| `l_instr` | liste d'instructions |
| `prog` | programme |
| `varDec` | déclaration de variable |
| `var_simple` | variable simple |

(2pt)

```
<prog>
  <l_dec>
    <foncDec>
    carre
      <l_dec>
        <varDec>$x</varDec>
      </l_dec>
      <l_instr>
        <instr_retour>
          <opExp>
            fois
            <varExp>
              <var_simple>$x</var_simple>
            </varExp>
            <varExp>
              <var_simple>$x</var_simple>
            </varExp>
          </opExp>
        </instr_retour>
      </l_instr>
    </foncDec>
    <l_dec>
      <foncDec>
        main
        <l_dec>
          <varDec>$x</varDec>
        </l_dec>
        <l_instr>
          <instr_affect>
            <var_simple>$x</var_simple>
            <intExp>2</intExp>
```

```
          </instr_affect>
          <l_instr>
            <instr_appel>
              <appel>
                carre
                <l_exp>
                  <varExp>
                    <var_simple>$x</var_simple>
                  </varExp>
                  <l_exp>
                  </l_exp>
                </l_exp>
              </appel>
            </instr_appel>
          </l_instr>
        </l_instr>
      </foncDec>
    </l_dec>
  </l_dec>
</prog>
```

**Q3.2** Écrire le programme en assembleur X86 que générerait votre compilateur pour le programme en L ci-dessus. Les instructions doivent comporter des commentaires expliquant leur rôle. (3pt)

```
%include 'io.asm'
section .bss
sinput: resb 255 ; reserve a 255 byte space in memory for the users input string
section .text
global _start
_start:
call main
mov eax, 1  ; 1 est le code de SYS_EXIT
int 0x80  ; exit
carre:
push ebp  ; sauvegarde la valeur de ebp
mov ebp, esp  ; nouvelle valeur de ebp
mov ebx, [ebp + 8]  ; lit variable dans ebx
push ebx
mov ebx, [ebp + 8]  ; lit variable dans ebx
push ebx
pop ebx  ; depile la seconde operande dans ebx
pop eax  ; depile la permière operande dans eax
imul ebx  ; effectue l'opération
push eax  ; empile le résultat
pop eax
mov [ebp + 12], eax  ; ecriture de la valeur de retour
pop ebp  ; restaure la valeur de ebp
ret
pop ebp  ; restaure la valeur de ebp
ret
```

```
main:
push ebp  ; sauvegarde la valeur de ebp
mov ebp, esp  ; nouvelle valeur de ebp
sub esp, 4 ; allocation variables locales
push 2
pop ebx
mov [ebp - 4], ebx  ; stocke registre dans variable
sub esp, 4  ; allocation valeur de retour
; empile arg 0
mov ebx, [ebp - 4]  ; lit variable dans ebx
push ebx
call carre
add esp, 4 ; desallocation parametres
add esp, 4  ; valeur de retour ignoree
add esp, 4 ; desallocation variables locales
pop ebp  ; restaure la valeur de ebp
ret
```

Voici les premières lignes du code généré :

```
%include 'io.asm'
section .bss
sinput: resb 255   ; alloue 255 octets pour une chaîne de caractèrtes entrée
section .text
global _start
_start:
    call   main    ; exécute la fonction main
    mov    eax, 1  ; 1 est le code de SYS_EXIT
    int    0x80    ; exit
```

Voici la liste des registres que vous avez besoin de connaître :

| | |
|---|---|
| `esp` | adresse du sommet de pile |
| `ebp` | adresse de base de la trame d'activation |
| `eip` | adresse de la prochaine instruction à exécuter |
| `eax, ebx, ecx, edx` | registres généraux |

Et voici la liste (minimale) des instructions dont vous avez besoin :

| | | |
|---|---|---|
| `add` | `dest, src` | effectue `dest = dest + src` |
| `sub` | `dest, src` | effectue `dest = dest - src` |
| `imul` | `src` | effectue `eax = eax * src` |
| `mov` | `dest, src` | copie `src` dans `dest` |
| `push` | `src` | copie `src` au sommet de la pile et décrémente `esp` |
| `pop` | `dest` | copie 4 octets du sommet de la pile dans `dest` et incrémente `esp` |
| `call` | `adr` | empile `eip` et va à l'adresse `adr` |
| `ret` | | dépile le sommet de la pile dans `eip` |