Final – LO42
solution

Les documents ne sont pas autorisés (La copie ou les idées du voisin non plus). Le barème est indicatif.
Le soin donné à la rédaction sera évalué. Toute réponse devra être claire et justifiée (toute ambiguïté sera mal interprétée). L'élégance de la solution sera jugée.
Sauf indication contraire, dans le cas d'algorithmes les réponses doivent être rédigées en pseudo code.

1) Les fonctions sont pâles ! (6)

Voici les descriptions fonctionnelles des opérations "val_min" et "val_max" d'un arbre binaire de recherche

Valmin (Arbre_vide) indéfini
Valmin (< n, G, D >) = Si G = nul Alors n
Sinon Valmin(G) ;
Valmax (Arbre_vide) indéfini
Valmax (< n, G, D >) = Si D = nul Alors n
Sinon Valmax (D) ;

Donnez les descriptions fonctionnelles des opérations :

- **hauteur** qui donne le nombre de niveaux d'un arbre binaire ;
- **nbéléments** qui donne le nombre d'éléments contenus dans un arbre ;
- **complet** qui vérifie si un arbre binaire est complet c'est à dire que toutes les feuilles soient sur un même niveau ou encore que le dernier niveau est complet ;
- **arbre_parfait** qui vérifie qu'un arbre est parfait ;

Solution :

hauteur(arbre_vide) = 0 ;
hauteur(<n, G, D>) = max(hauteur(G), hauteur(D)) + 1

nbéléments(arbre_vide) = 0 ;
nbéléments(<n, G, D>) = nbéléments(G) + nbéléments(D) + 1

complet(arbre_vide) = vrai ;
complet(<n, G, D>) = (hauteur (G) = hauteur (D)) et complet(G) et complet(D)
ou
complet(<n, G, D>) = (nbéléments (G) = nbéléments (D)) et complet(G) et complet(D)

arbre_parfait(arbre_vide) = vrai ;
arbre_parfait(<n, G, D>) = ((hauteur(G) = hauteur(D)) et complet (G) et arbre_parfait (D))
ou (hauteur(G) = hauteur(D) + 1) et arbre_parfait (G) et complet (D))

2) Aujourd'hui, je montre le fond de la chose (7)

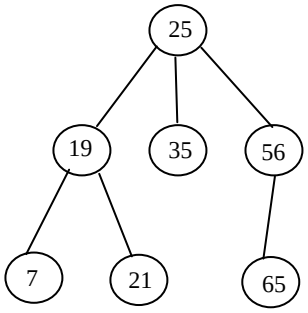
Voici un algorithme récursif de parcours en profondeur d'une arborescence.

```
Procédure Parcours (n : nœud) ;
Début
        Tantque n <> nul Alors
                parcours (fils_gauche(n)) ;
                n ← frère_droit(n) ;
        Ftq ;
fin ;
```

a) Ecrivez une procédure toute récursive en éliminant l'itération du parcours en profondeur.

b) Modifiez cette procédure toute récursive, en introduisant des appels aux traitements en pré-ordre, en infixé et en post-ordre (pre(x), inf(x), post(x)). Les exécutions des traitements doivent suivre les règles définies.
En préliminaire, sur l'exemple ci-dessous étudiez ce que fait chaque appel.

c) Inspirez vous de ces résultats pour afficher, à l'aide d'une procédure tout récursive, la structure d'une arborescence sous forme textuelle comme ci-dessous.



25 (19 (7, 21), 35, 56 (65))

Vous utiliserez une opération "**écrire**" pour afficher des valeurs ou des chaînes de caractères. L'opération "**val**" vous donnera la valeur associée à un nœud.

Solution :

a) On voit que la procédure correspond au schéma itératif d'une procédure de type I. On procède donc à l'opération inverse.

```
Procédure parcours (n : nœud) ;
Début
        Si n <> nul Alors
                parcours( fils_gauche(n) ) ;
                parcours( frère_droit(n) ) ;
        Fsi ;
fin ;
```

b) L'étude de l'exemple nous montre que l'appel du parcours avec fils gauche de n va exécuter tout les parcours au niveau des fils de n. Le deuxième appel sert à l'exploration des frères de n.
Pour le traitement infixé, il est nécessaire de transmettre la valeur ou le nœud père puisque les parcours de toutes les sous arborescences liées à un nœud sont faits dans le premier appel. Le passage du fils gauche au premier fils droit se fait dans cet appel. Toutefois si le sommet n'a pas de fils, le traitement ne sera pas fait dans le parcours du fils gauche. Il faut réaliser le traitement en dehors de l'appel. Un autre solution est d'intégrer le fait que si un fils gauche est nul il faut traiter le père.

```
Procédure parcours (n : nœud) ;
Début
    parcoursb (n, nul ) ;
fin ;

Procédure parcoursb (n, père : nœud) ;
Début
    Si n <> nul Alors
        pre( val(n) ) ;
        Si fils_gauche(n) = nul Alors inf(n) ;
        Sinon parcoursb( fils_gauche(n), n ) ; Fsi ;
        post( val(n) ) ;
        si père<> nul Alors inf(val(père)) ;
        Fsi ;
        parcoursb( frère_droit(n), nul ) ;
    Fsi ;
fin ;
```

Ou pour éviter le test sur le fils gauche qui est réalisé par le test global au niveau suivant des appels, on reporte le inf(n) dans l'appel dans la clause sinon du test global.

```
Procédure parcoursb (n, père : nœud) ;
Début
    Si n <> nul Alors
        pre( val(n) ) ;
        parcoursb( fils_gauche(n), n ) ;
        post( val(n) ) ;
        si père<> nul Alors inf(val(père)) ; Fsi ;
        parcoursb( frère_droit(n), nul ) ;
    Sinon si père<> nul Alors inf(val(père)) Fsi ;
    Fsi ;
fin ;
```

c) Pour l'affichage il y a un traitement en pré-ordre, un traitement en post-ordre et un traitement qui se fait entre chaque fils. Ce dernier n'a pas besoin de l'information liée au père, on peut donc éviter de le transmettre.
La valeur doit être traitée en pré-ordre, ainsi que la parenthèse ouvrante.
La virgule est imprimé entre chaque fils
La parenthèse fermante est imprimée en post ordre.

De plus comme les parenthèses ne doivent pas être imprimées si le nœud est une feuille, on introduit un test. Même raisonnement pour le dernier fils.

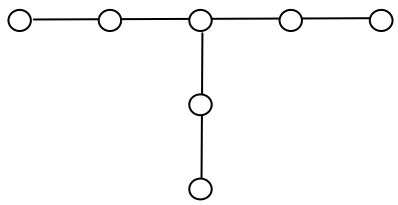
```
Procédure affich (n : nœud) ;
Début
        Si n <> nul Alors
                ecrire( val(n) ) ;
```

```
                    Si fils_gauche(n) <> nul Alors
                        ecrire("( ") ;
                        affich( fils_gauche(n) ) ;
                        ecrire(" )") ;
                    Fsi ;
                    si frère_droit(n) <> nul Alors
                        ecrire(",  ") ;
                        affich( frère_droit(n) ) ;
                    Fsi ;
                Fsi ;
        fin ;
```

3) Et on redémarre ! (7)

Un caterpillar est un arbre ayant une chaîne (élémentaire) contenant au moins un sommet de chaque arête. Une telle chaîne est dite dominante.

1. Montrez que, si T est un arbre, les trois conditions suivantes sont équivalentes :
   a. T est un caterpillar ;
   b. Tout sommet de T a au plus deux voisins qui ne sont pas des feuilles ;
   c. T ne contient pas le sous-graphe ci-dessous.



2. Donnez un algorithme détaillé en O(n) pour reconnaître si un arbre est un caterpillar. Justifiez la méthode.
   On utilisera le fait qu'un arbre est un graphe et permet, par conséquent, les opérations sur graphe.

Solution :

1. Remarque : Pour prouver l'équivalence des conditions il fait faire la preuve dans les deux sens.
   $a \Rightarrow b$ et $b \Rightarrow a$ nous permet de dire $a \Leftrightarrow b$
   On peut faire également $a \Rightarrow b \Rightarrow c \Rightarrow a$

Commençons par démontrer $b \Leftrightarrow c$

On veut démontrer $c \Rightarrow b$ par $\neg b \Rightarrow \neg c$

Si un sommet à plus de trois voisins qui ne sont pas des feuilles alors ce sommet peut être représenté comme sommet central du sous graphe défini en c. Donc le sous-graphe existe dans l'arbre.

Inversement si le sous-graphe existe dans l'arbre alors il existe un sommet de l'arbre qui a plus de deux voisins qui ne sont pas des feuilles. On a démontré $\neg c \Rightarrow \neg b$ donc $b \Rightarrow c$.

$a \Rightarrow c$ par $\neg c \Rightarrow \neg a$

Si l 'arbre T contient un sous graphe comme défini dessus, il n'existe pas de chaîne élémentaire passant par les quatre sommets centraux. T étant un arbre il n'y a pas de cycle, Donc une chaîne peut passer par trois des sommets mais pas les quatre. De plus, l'arête extérieure liée au sommet n'appartenant pas à la chaîne ne peut être adjacente à cette chaîne par la non-existence de cycle. Il n'y a donc pas de chaîne élémentaire contenant au moins un sommet de chaque arête.

$b \Rightarrow a$ par $\neg a \Rightarrow \neg b$

S'il n'existe pas de chaîne élémentaire contenant au moins un sommet de chaque arête, il existe une arête dont aucun sommet n'appartient à la chaîne élémentaire. T étant un arbre un des sommets de l'arête est relié à la chaîne élémentaire. La liaison ne peut pas être sur un sommet extrémité de la chaîne sinon il suffirait de prolonger la chaîne pour contenir un sommet de chaque arête. La liaison à la chaîne ne peut se faire que sur un sommet qui a déjà deux voisins dans la chaîne élémentaire. Ce sommet a donc au minimum 3 voisins qui ne sont pas des feuilles (les deux voisins dans la chaîne plus le sommet par lequel est liée l'arête indépendante).

$a \Rightarrow b$

Soit s un sommet d'un caterpillar T, il y a deux cas possibles :

s est un sommet ne possédant qu'un voisin (c'est une feuille),

s est un sommet possédant plus d'un voisin. Comme T est un arbre, la chaîne dominante doit passer par s pour être en relation avec toutes les arêtes.

Dans une chaîne élémentaire un sommet a au plus deux sommets adjacents ou voisins.

Comme un sommet possédant plus d'un voisin, fait partie de la chaîne dominante élémentaire, il ne peut posséder plus de deux voisins dans cette chaîne élémentaire, c'est à dire qui ne sont pas des feuilles.

$b \Rightarrow a$

T est un arbre donc connexe et sans cycle. On peut donc passer une chaîne par tous les sommets de T qui ne sont pas des feuilles. Tout sommet a au plus 2 sommets voisins qui ne sont pas des feuilles. Si la chaîne passe par le sommet s, alors soit elle s'arête sur s, soit elle passe par s pour relier ses deux voisins et ne peut revenir sur s du fait de l'absence de cycle. La chaîne est donc élémentaire. T étant un arbre tout sommet constituant une feuille est relié à un sommet qui n'en est pas une. La chaîne élémentaire passant par tous les sommets qui ne sont pas des feuilles contient donc un sommet de chaque arête.

2. Nous avons un arbre, donc le graphe est connexe.
   Chaque sommet a au plus 2 sommets voisins qui ne sont pas des feuilles. Pour chaque successeur d'un sommet on vérifie si celui-ci est une feuille ou non. Si ce successeur n'est pas une feuille on le comptabilise. A la fin de la liste des successeurs d'un sommet si le compte est supérieur à 2, l'arbre n'est pas un caterpillar.

On parcourt la liste des sommets par la liste principale.

```
Fonction caterpillar(B : Arbre) : Booléen ;
Var   ok : Booléen ;
      i, j, nb_voi_noeud : Entier ;
Début
      ok ← vrai ;
      i ←1 ;
      Tantque  i < Ordre(B) et ok Faire
            nb_voi_noeud ← 0 ;
            Pour j de 1 à degré(i) Faire
                  Si degré( ième_succ(j, i) ) > 1 Alors
                        nb_voi_noeud ++ ;
                  Fsi ;
            Finpour ;
            ok ← nb_voi_noeud < 3 ;
            i++ ;
      Ftq ;
      caterpillar ← ok ;
Fin ;
```

Une autre solution est de parcourir l'arbre à partir d'un sommet.

```
Fonction caterpillar(B : Arbre ; s, père : entier) :Booléen ;
Var   ok : Booléen ;
      i, j, nb_voi_noeud : Entier ;
Début
      ok ← vrai ;
      i ←1 ;
      nb_voi_noeud ← 0 ;
      Tantque  ok et i < degré(s) Faire
            j ← ième_succ(i, s) ;
            Si degré(j ) > 1 Alors
            Si j <> père Alors ok ← caterpillar(B, j, s) ; Fsi ;
                  nb_voi_noeud ++ ;
            Fsi ;
            ok ← nb_voi_noeud < 3 et ok ;
            i++ ;
      Ftq ;
      caterpillar ← ok ;
Fin ;
```