AL 4I - Algorithmique avancée - Contrôle continu

Nom : _____

Prénom : _____

- Durée : 1 heure. Documents interdits, calculatrices interdites.
- Écrivez toutes les réponses directement sur le sujet. Si vous n'avez pas suffisamment de place, écrivez au dos d'une feuille en le **précisant dans la question**.
- La dernière page du sujet vous donne les notations à utiliser pour rédiger les algorithmes. Lisez-là avant de commencer.
- Il est interdit d'utiliser des boucles, seuls les algorithmes récursifs sont autorisés.

1 Listes chaînées (6 points)

1. [2 points] Ecrire une fonction $nbMaillons(l)$ retournant le nombre de maillons de la liste l.

Solution:

```
Fonction nbMaillons(l)
    si l = null alors
        retourner 0
    sinon
        retourner 1 + nbMaillons(l.suivant)
    fin
FIN
```

2. [2 points] Ecrire une fonction $copie(l)$ retournant une copie de la liste l.

Solution:

```
Fonction copie(l)
    si l = null alors
        retourner null
    sinon
        retourner LC(l.donnee, copie(l.suivant))
    fin
FIN
```

3. [2 points] Ecrire une fonction $inverse(l)$ retournant la liste l dont l'ordre des maillons a été inversé, vous créerez pour l'occasion une fonction récursive terminale.

Solution:

```
Fonction inverseAcc(l, a)
    si l = null alors
        retourner a
    sinon
        deuxieme ⟵ l.suivant
        l.suivant ⟵ a
        retourner inverseAcc(deuxieme, l)
    fin
FIN

Fonction inverse(l)
    retourner inverseAcc(l, null)
FIN
```

2 Arbres binaires (8 points)

1. [2 points] Ecrire une fonction $somme(a)$ retournant la somme des données des noeuds de l'arbre a.

Solution:

```
Fonction somme(a)
    si a = null alors
        retourner 0
    sinon
        retourner
        a.donnee + somme(a.gauche) + somme(a.droite)
    fin
FIN
```

2. [2 points] Ecrire une fonction $contient(a, x)$ retournant vrai si et seulement si un des noeuds de l'arbre a contient la donnée x.

Solution:

```
Fonction contient(a)
    si a = null alors
        retourner faux
    sinon
        retourner (a.donnee =
        x) ∨ contient(a.gauche, x) ∨ contient(a.droite, x)
    fin
FIN
```

3. 2 points Ecrire une fonction $copieNoeuds(a)$ retournant une copie de l'arbre a sans les feuilles.

Solution:

```
Fonction copieNoeuds(a)
    si a = null ou (a.gauche = null et a.droite = null)
    alors
        retourner null
    sinon
        retourner
        AB(copieNoeuds(a.gauche), a.donnee, copieNoeuds(a.droite))
    fin
FIN
```

4. 2 points Ecrire une fonction $enListe(a)$ retournant une liste contenant les données de a. Vous créerez pour l'occasion une fonction récursive terminale.

Solution:

```
Fonction enListeAcc(a, acc)
    si a = null alors
        retourner acc
    sinon
        acc ⟵ enListeAcc(a.droite, acc)
        acc ⟵ LC(a.donnee, acc)
        retourner enListeAcc(a.gauche, acc)
    fin
FIN

Fonction enListe(a)
    retourner enListeAcc(a, null)
FIN
```

3 Arbres (6 points)

1. 3 points Ecrire une fonction $sommeArbre(a)$ retournant la somme des donnée de l'arbre a. Vous créerez pour l'occasion une fonction $sommeFils(f)$ mutuellement récursive avec $sommeArbre$.

Solution:

```
Fonction sommeArbre(a)
    si a = null alors
        retourner 0
    sinon
        retourner a.donnee + sommeFils(a.fils)
    fin
FIN

Fonction sommeFils(f)
    si f = null alors
        retourner 0
    sinon
        retourner
        sommeArbre(f.donnee) + sommeFils(f.suivant)
    fin
FIN
```

2. 3 points Ecrire une fonction $enArbre(a)$ retournant un arbre contenant les mêmes données et ayant la même structure que l'arbre **binaire** a.

Solution:

```
Fonction enArbre(a)
    si a = null alors
        retourner null
    sinon
        retourner
        A(a.donnee, LC(enArbre(a.gauche), LC(enArbre(a.droit), null)))
    fin
FIN
```

4 Notations

De façon générale, on ne se souciera pas des problèmes de libération de la mémoire.

4.1 Listes chaînées

Nous gérerons les listes avec une structure de type LC comportant deux champs *donnee* et *suivant* auxquels on accèdera en utilisant une notation pointée. Si par exemple l est de type LC, alors $l.donnee$ est la donnée et $l.suivant$ le maillon suivant de la liste. La liste vide sera notée *null* et le constructeur $LC(d, s)$ retourne un maillon contenant la donnée d et dont l'adresse du successeur est s.

4.2 Arbres binaires

Nous gérerons les arbres binaires avec une structure de type AB comportant trois champs *donnee*, *gauche* et *droite*. Si par exemple a est de type AB, alors $a.donnee$ est la donnée, $a.gauche$ le sous-arbre gauche et $a.droite$ le sous-arbre droit. L'arbre vide sera notée *null* et le constructeur $AB(g, d, dr)$ retourne un noeud contenant la donnée d, ayant g pour sous-arbre gauche et d pour sous-arbre droit.

4.3 Arbres

Nous gérerons les arbres avec une structure de type A comportant deux champs *donnee* et $fils$. Si par exemple a est de type A, alors $a.donnee$ est la donnée, $a.fils$ la liste des fils. L'arbre vide sera notée *null* et le constructeur $A(d, f)$ retourne un noeud contenant la donnée d et ayant f comme liste de fils.