2013-2014	L'3 – UE « Structures de données » (SDD) Devoir écrit final Mercredi 08 janvier 2014 Durée 2h00 – Sans document ni équipement électronique	A. Morelle

Corrigé

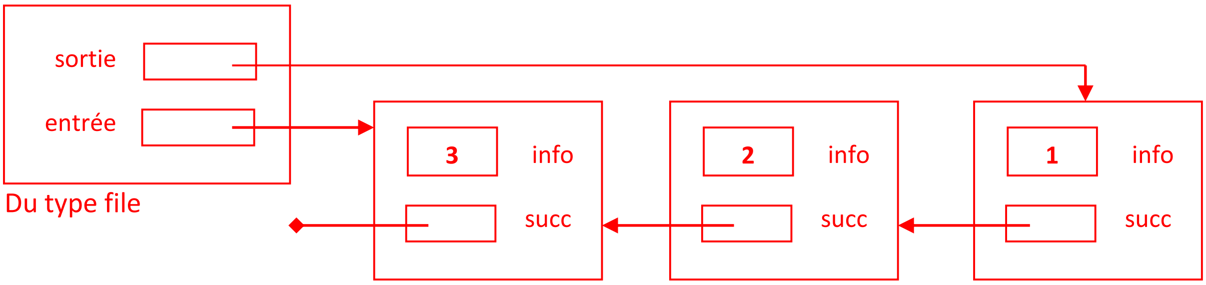
Les algorithmes demandés sont typiquement attendus en langage algorithmique et doivent être autonomes (pas d'appel à des algorithmes de sous-traitance). Les solutions seront également appréciées au regard de leur efficacité (par exemple ne pas réaliser deux parcours d'un arbre quand un seul suffirait).

Exercice 1 [3 points]

1.1-Spécifier (en langage algorithmique ou en C) les types de données permettant de définir une file d'attente d'entiers à capacité non bornée a priori.

1.2-Dessiner la représentation mémoire complète (maillons, champs, chaînages, …) d'une file d'attente de 3 valeurs : 1, 2 et 3, supposées arrivées en file dans cet ordre.

1.3-Ecrire une procédure *enfiler(f, n)* qui met en file f une valeur n.



Procédure enfiler(f : adresse file, n : entier)
// Variable locale : pointeur p de type adresse maillon
Début

p ← réserver maillon
p→info ← n
p→succ ← null
si f→entrée = null // ou f→sortie = null
alors f→sortie ← p
sinon f→entree→succ ← p
finsi
f→entrée ← p

Fin

Exercice 2 [3 points]

Ecrire une procédure itérative répondant à la spécification suivante :

Procédure insérer(a : ABR, n : entier)
// Donnée modifiée : un arbre binaire de recherche a ; l'ABR modifié final est l'arbre initial dans
// lequel a été inséré un nœud de valeur n
// Donnée : la valeur n à insérer dans l'ABR a
// Variables locales : pointeurs p et pp de type adresse noeud
Début

```
        p  ← a
        pp ← null       // père de p
        tant que p ≠ null
             pp ← p
             si n ≤ p→info
                 alors p ← p→ sag
                 sinon p ← p→ sad
             finsi
        finTantQue

        p ← réserver nœud
        p→info ← n
        p→sag ← null
        p→sad ← null

        si pp = null alors a ← p
        sinon  si n ≤ pp→info alors pp→sag ← p
        sinon  pp→sad ← p
```

Fin

Exercice 3 [3 points]

Ecrire une procédure répondant à la spécification suivante :

Procédure détruire(a : AB)
// Donnée modifiée : un arbre binaire quelconque a à détruire (chacun de ses nœuds sera libéré) ;
// l'arbre final est un arbre vide.
// Variables locales : pointeurs p et pp de type adresse noeud
Début

```
        Si a = null alors retourner ;
        // sinon
        détruire(a→sag)
        détruire(a→sad)
        libérer a
        a ← null
```

Fin

Exercice 4 [3 points]

Ecrire une fonction répondant à la spécification suivante :

```
Fonction moyenne(a : AB, n : EntierNat) : réel
// Donnée : un arbre binaire quelconque a à évaluer
// Donnée modifiée : un entier naturel n dont la valeur finale est le nombre de nœuds de l'arbre a
// Résultat : la valeur moyenne des valeurs contenues dans l'arbre a, ou 0 si l'arbre est vide.
// Variables locales : réels mg, md ; entiers ng, nd.
Début
        Si  a = null   alors
            n ←  0
            retourner 0
        finsi
        // sinon
        mg ← moyenne(a→sad, ng)
        md ← moyenne(a→sad, nd)
        n  ← ng + nd + 1
        retourner ( ng * mg  +  nd * md  +  a→info ) / n
Fin
```

Exercice 5 [4 points]

On adopte la définition suivante : « un arbre binaire est parfait ssi il est complet (càd tous ses noeuds internes sont de degré deux) et toutes ses feuilles sont à la même profondeur ».

5.1- Dessiner un arbre binaire parfait de hauteur 2 dont les valeurs des nœuds sont telles qu'un affichage de l'arbre par un parcours en profondeur post-ordre (post-fixe) afficherait : 1, 2, 3, 4, 5, 6, 7.

```
        7
      /   \
     3     6
    / \   / \
   1   2 4   5
```

5.2- Ecrire une fonction qui crée et retourne un arbre binaire parfait de hauteur h dont les valeurs des nœuds sont telles qu'un affichage de l'arbre par un parcours en profondeur post-ordre (post-fixe) afficherait toutes les valeurs entières de 1 à 2^{h+1}-1 inclus.

```
Fonction creerABP(h : entier, n : entier) : arbre
// Donnée : hauteur h de l'arbre à construire
// Donnée : valeur du nœud racine (2^(h+1)-1 au premier appel)
// Résultat : l'AB parfait … répondant à l'énoncé
// Variable locale : une variable a de type AB
Début
        Si h < 0 alors retourner NULL
        a ← réserver nœud
        a→info ←  n
        a→sag ←  creerABP(h – 1, n – 2^h)
        a→sad ←  creerABP(h – 1, n – 1)
        retourner a
Fin
```

Exercice 6   [4 points]

En s'inspirant de la méthode du Quick Sort -- dont on pourra notamment utiliser telle quelle, sans la réécrire, la sous-fonction partitionner mais *en indiquant toutefois précisément ce qu'elle produit* -- et *sans trier totalement le tableau par ordre croissant ou décroissant* !! (car inutile), écrire un algorithme efficace qui détermine la valeur médiane d'un tableau t de n valeurs (n supposé impair) indicé de 0 à n-1.

Exemple. Le tableau

0	1	2	3	4	5	6	7	8
6	3	5	7	2	8	1	1	7

a pour valeur médiane 5

```
Fonction médiane(t : tableau d'entiers, i : entier, j : entier, m : entier) : entier
// Premier appel pour un tableau t[0..n-1] : médiane(t, 0, n-1, (n-1)/2)
// Donnée modifiée : le tableau t[i..j]. Le tableau final t[i..j] est une permutation du tableau initial.
//                        Propriété du tableau final : t[i..m-1] ≤ t[m] < t[m+1..j] et t[m] a pour valeur la
//                        médiane de t[i..j]
// Résultat : la valeur médiane de t[i..j]
// Variable locale : un entier k
Début
        Si i = j alors retourner t[i]
        k ← partitionner(t, i, j)
        si  m < k  alors retourner mediane(t, i, k-1, m)
        si  m > k  alors retourner mediane(t, k+1, j, m)
        retourner t[k]
Fin
```