ECOLE SUPÉRIEURE POLYTECHNIQUE

DEPARTEMENT GENIE INFORMATIQUE

Filaire : Informatique

Matière : Algorithmique et programmation avancées

FICHE A3 : LISTE CHAINEE

Membres :

Mohamed MBAYE

Babacar NDIAYE

Nerva-Kelen SONCY

Mohamed Sall

Abdoulaye Barro

Korka Sall

Cheikh Touradou Lassana Gueye

Mouhamed AMAR

EXERCICE 1 :

```
class Maillon {
    String mot;
    Maillon suivant;
    public Maillon(String mot) {
        this.mot = mot;
        this.suivant = null;
    }
}

public class ListeChainee {
    // 1. Ajouter un mot au début de la liste
    public static Maillon ajouteDebut(String mot, Maillon L) {
        Maillon nouveau = new Maillon(mot);
        nouveau.suivant = L;
        return nouveau;
    }
```

// 2. Ajouter un mot à la fin de la liste

```
public static Maillon ajouteFin(String mot, Maillon L) {
    Maillon nouveau = new Maillon(mot);
    if (L == null) return nouveau;
    Maillon courant = L;
    while (courant.suivant != null) {
        courant = courant.suivant;
    }
    courant.suivant = nouveau;
    return L;
}
```

// 3. Supprimer un mot

```
public static Maillon supprimer(String mot, Maillon L) {
    if (L == null) return null;
    if (L.mot.equals(mot)) {
        return L.suivant;
    }
```

```
    Maillon courant = L;
    while (courant.suivant != null && !courant.suivant.mot.equals(mot)) {
        courant = courant.suivant;
    }
    if (courant.suivant != null) {
        courant.suivant = courant.suivant.suivant;
    }
    return L;
}
```

// 4. Afficher les n premiers mots

```
public static void premiers(Maillon L, int n) {
    int compteur = 0;
    Maillon courant = L;
    while (courant != null && compteur < n) {
        System.out.println(courant.mot);
        courant = courant.suivant;
        compteur++;
    }
```

```
}
```

// 5. Purifier la liste triée en supprimant les doublons

```
public static void purifie(Maillon L) {
    Maillon courant = L;
    while (courant != null && courant.suivant != null) {
        if (courant.mot.equals(courant.suivant.mot)) {
            courant.suivant = courant.suivant.suivant;
        } else {
            courant = courant.suivant;
        }
    }
}

public static void afficherListe(Maillon L) {
    Maillon courant = L;
    while (courant != null) {
        System.out.print(courant.mot + " -> ");
        courant = courant.suivant;
```

```
    }
    System.out.println("null");
}

public static void main(String[] args) {
    Maillon liste = null;
    liste = ajouteFin("apple", liste);
    liste = ajouteFin("banana", liste);
    liste = ajouteFin("banana", liste);
    liste = ajouteFin("cherry", liste);
    liste = ajouteFin("date", liste);
    System.out.println("Liste initiale :");
    afficherListe(liste);
    System.out.println("\nSuppression de 'banana' :");
    liste = supprimer("banana", liste);
    afficherListe(liste);
    System.out.println("\nAjout de 'kiwi' au début :");
    liste = ajouteDebut("kiwi", liste);
    afficherListe(liste);
```

```
        System.out.println("\nAffichage des 3 premiers mots :");

        premiers(liste, 3);

        System.out.println("\nListe après purification (triée avec doublons supprimés) :");

        purifie(liste);

        afficherListe(liste);

    }

}
```

EXERCICE 2 : Listes chaînées bidirectionnelles de chaînes de caractères

1) Déclaration de la structure maillon et des variables globales premier et dernier :

```
class MaillonBi {
        String val;
        MaillonBi prec;
        MaillonBi suiv;

        public MaillonBi(String val) {
            this.val = val;
        }
}

class LCB {
        MaillonBi premier;
```

```
MaillonBi dernier;
```

2) Fonction ajouter_devant(char *s) – version Java avec variables globales :

```
public void ajouterDevant(String val) {
MaillonBi nouveau = new MaillonBi(val);
nouveau.suiv = premier;

if (premier != null) {
   premier.prec = nouveau;
} else {
   dernier = nouveau;
}

premier = nouveau;
}
```

3) Réécriture de ajouter_devant avec premier et dernier passés en paramètres :

```
// Cette version retourne un tableau contenant les nouveaux pointeurs [premier, dernier]

public class UtilsLCB {

    public static MaillonBi[] ajouterDevant(String val, MaillonBi premier, MaillonBi dernier) {

    MaillonBi nouveau = new MaillonBi(val);

    nouveau.suiv = premier;
```

```
        if (premier != null) {
           premier.prec = nouveau;
        } else {
           dernier = nouveau;
        }

        premier = nouveau;

        return new MaillonBi[] { premier, dernier };
        }
}
```

4) Fonction supprimer(char *s) – suppression du premier maillon contenant s :

```
        public void supprimer(String val) {
        MaillonBi courant = premier;



        while (courant != null && !courant.val.equals(val)) {
           courant = courant.suiv;
        }

        if (courant == null) return;

        if (courant.prec != null) {
           courant.prec.suiv = courant.suiv;
        } else {
           premier = courant.suiv;
```

```
        }

        if (courant.suiv != null) {
           courant.suiv.prec = courant.prec;
        } else {
           dernier = courant.prec;
        }
        }
}
```

EXERCICE 3 :

```
import java.util.Scanner;

class Node {

    int data;

    Node next;

    Node(int data) {

        this.data = data;

        this.next = null;

    }

}

public class ListeChainee {

    public static Node insertEnd(Node head, int val) {

        Node newNode = new Node(val);

        if (head == null) {

            return newNode;

        }

        Node current = head;
```

```
    while (current.next != null) {
        current = current.next;
    }
    current.next = newNode;
    return head;
}
// Question 1 : Créer une liste de 10 valeurs saisies au clavier.
public static Node createList(int n) {
    Scanner scanner = new Scanner(System.in);
    Node head = null;
    for (int i = 0; i < n; i++) {
        System.out.print("Saisir la valeur " + (i + 1) + " : ");
        int val = scanner.nextInt();
        head = insertEnd(head, val);
    }
    return head;
}
public static void printList(Node head) {
```

```
        Node current = head;
       while (current != null) {
            System.out.print(current.data + " -> ");
            current = current.next;
        }
        System.out.println("NULL");
    }
    // Question 2 : Tester l'égalité de deux listes.
public static boolean compareLists(Node l1, Node l2) {
        while (l1 != null && l2 != null) {
            if (l1.data != l2.data)
                return false;
            l1 = l1.next;
            l2 = l2.next;
        }
        return (l1 == null && l2 == null);
    }
    // Question 3: Concaténer deux listes
```

```
public static Node concatNewList(Node l1, Node l2) {
    Node newHead = null;
    Node temp = l1;
    while (temp != null) {
        newHead = insertEnd(newHead, temp.data);
        temp = temp.next;
    }
    temp = l2;
    while (temp != null) {
        newHead = insertEnd(newHead, temp.data);
        temp = temp.next;
    }
    return newHead;
}
```

```
// Question 3b : Concaténer deux listes sans créer de troisième liste
public static Node concatInPlace(Node l1, Node l2) {
```

```
        if (l1 == null)
            return l2;
        Node current = l1;
        while (current.next != null)
            current = current.next;
        current.next = l2;
        return l1;
    }
```

Méthode principale (main) qui orchestre l'exécution de l'exercice.

```
    public static void main(String[] args) {
        System.out.println("Création de la première liste (10 valeurs) :");
        Node liste1 = createList(10)
        System.out.println("\nCréation de la deuxième liste (10 valeurs) :");
        Node liste2 = createList(10)
        System.out.print("\nListe 1 : ");
        printList(liste1);
        System.out.print("Liste 2 : ");
        printList(liste2);
```

```
        if (compareLists(liste1, liste2))
            System.out.println("\nLes deux listes sont égales.");
        else
            System.out.println("\nLes deux listes ne sont pas égales.");
         System.out.println("\nConcaténation (création d’une troisième liste) :");
        Node liste3 = concatNewList(liste1, liste2);
        System.out.print("Liste 3 (liste1 + liste2) : ");
        printList(liste3)
        System.out.println("\nConcaténation in-place (liste1 = liste1 + liste2) :");
        liste1 = concatInPlace(liste1, liste2);
        System.out.print("Liste 1 après concaténation in-place : ");
        printList(liste1);
    }
}
```

EXERCICE 4 : Dérivée d'un polynôme

```
public class Polynome {

    // 1°) Structure pour représenter un polynôme par une liste chaînée

    static class Terme {

        int coef;

        int expo;

        Terme suivant;



        Terme(int coef, int expo) {

            this.coef = coef;

            this.expo = expo;

            this.suivant = null;

        }

    }

    Terme tete = null;

    // Ajouter un terme (utile pour créer le polynôme facilement)

    public void ajouterTerme(int coef, int expo) {
```

```
        if (coef == 0) return;
        Terme nouveau = new Terme(coef, expo);
        nouveau.suivant = tete;
        tete = nouveau;
    }
    // Affichage du polynôme
    public void afficher() {
        Terme courant = tete;
        while (courant != null) {
            System.out.print(courant.coef + "x^" + courant.expo);
            if (courant.suivant != null) System.out.print(" + ");
            courant = courant.suivant;
        }
        System.out.println();
    }
    // 2°) Fonction DERIVEE(P) – calcule et retourne la dérivée première du polynôme
    public Polynome derivee() {
        Polynome d = new Polynome();
```

```
        Terme courant = tete;



        while (courant != null) {
            if (courant.expo != 0) {
                int newCoef = courant.coef * courant.expo;
                int newExpo = courant.expo - 1;
                d.ajouterTerme(newCoef, newExpo);
            }
            courant = courant.suivant;
        }



        return d;
    }
```

// 3°) Fonction DERIVEEKIEME(P, k) – calcule la dérivée k-ième

```
    public Polynome deriveeKiem(int k) {
        Polynome resultat = this;
        for (int i = 0; i < k; i++) {
            resultat = resultat.derivee();
```

```
        if (resultat.tete == null) break;
    }
    return resultat;
}
// Programme principal pour tester
public static void main(String[] args) {
    Polynome P = new Polynome();
    // On construit le polynôme : P(x) = 3x^3 + 2x^2 + 5
    P.ajouterTerme(5, 0);
    P.ajouterTerme(2, 2);
    P.ajouterTerme(3, 3);

    System.out.print("P(x) = ");
    P.afficher();
    // Affichage de la dérivée première : P'(x)
    Polynome P1 = P.derivee();
    System.out.print("P'(x) = ");
    P1.afficher();
```

```
        // Affichage de la dérivée seconde : P''(x)

        Polynome P2 = P.deriveeKiem(2);

        System.out.print("P''(x) = ");

        P2.afficher();

    }

}
```

EXERCICE 5 :

```
// 1. Déclarer en C des types MAILLON et PTR permettant de réaliser des
vecteurs creux.
class Maillon {
    int index;
    float valeur;
    Maillon suivant;
}

// 2. Ecrire une fonction nouveau_maillon permettant de créer un
nouveau maillon avec des
valeurs initiales fournies en arguments.
public static Maillon nouveauMaillon(int index, float valeur) {
    return new Maillon(index, valeur);
}

// 3. Ecrire une fonction PTR vecteur_creux (float t[ ], int n) qui prend un
tableau t ayant n
éléments et construit sa représentation sous forme de vecteur creux.
public static Maillon vecteurCreux(float[] t, int n) {
    Maillon tete = null, courant = null;

    for (int i = 0; i < n; i++) {
        if (t[i] != 0) {
            Maillon nouveau = new Maillon();
            nouveau.index = i;
            nouveau.valeur = t[i];
            nouveau.suivant = null;

            if (tete == null) {
                tete = nouveau;
                courant = nouveau;
            } else {
                courant.suivant = nouveau;
                courant = nouveau;
            }
        }
    }
    return tete;
}
```

```
// 4. Ecrire une fonction PTR somme (PTR a, PTR b) qui reçoit deux vecteurs creux a et b et
// retourne le vecteur creux qui représente leur somme (c'est-à-dire l'addition des deux
// vecteurs, composante par composante).

public static Maillon somme(Maillon a, Maillon b) {
    Maillon resultat = null;
    Maillon courant = null;

    while (a != null || b != null) {
        Maillon nouveau = null;

        if (a != null && (b == null || a.index < b.index)) {
            nouveau = nouveauMaillon(a.index, a.valeur);
            a = a.suivant;
        } else if (b != null && (a == null || b.index < a.index)) {
            nouveau = nouveauMaillon(b.index, b.valeur);
            b = b.suivant;
        } else if (a != null && b != null && a.index == b.index) {
            float somme = a.valeur + b.valeur;
            if (somme != 0) {
                nouveau = nouveauMaillon(a.index, somme);
            }
            a = a.suivant;
            b = b.suivant;
        }

        if (nouveau != null) {
            if (resultat == null) {
                resultat = nouveau;
                courant = nouveau;
            } else {
                courant.suivant = nouveau;
                courant = nouveau;
            }
        }
    }
    return resultat;
```

```
}
```

EXERCICE 6 : Matrices symétriques

```
// ============================

// EXERCICE 6 : MATRICES SYMETRIQUES

// ============================

public class MatriceCarree {

// QUESTION 1 : Déclarer un type MATCARREE avec un tableau 2D pour stocker la matrice

private double[][] mat; // La matrice complète (tableau 2D)

private int n; // Taille de la matrice (n x n)

// Constructeur : créer une matrice vide de taille n x n

public MatriceCarree(int n) {

this.n = n;

mat = new double[n][n]; // Allocation d'une matrice n x n remplie de 0.0 par défaut

}

// Méthode pour modifier un élément de la matrice

public void setVal(int i, int j, double val) {

mat[i][j] = val;

}

// Méthode pour lire un élément de la matrice

public double getVal(int i, int j) {

return mat[i][j];

}

public int getN() {

return this.n;

}

// ====================
```

```
// QUESTION 2 : Vérifier si la matrice est symétrique
// ====================
public boolean estSymetrique() {
// Parcours de toute la matrice
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
// Une matrice est symétrique si M[i][j] == M[j][i]
if (mat[i][j] != mat[j][i]) {
return false; // Dès qu'une inégalité est trouvée, ce n'est pas symétrique
}
}
}
return true; // Si aucune différence, la matrice est symétrique
}
// ======================
// QUESTION 3 : Combien d'éléments faut-il stocker ?
// ======================
// Réponse : n(n+1)/2 éléments
// C'est la taille de la partie triangulaire supérieure (diagonale incluse)
// Pas besoin d'écrire de code ici, c'est juste une formule mathématique.
// ======================
// QUESTION 4 : Générer la représentation compacte de la matrice
// ======================
public double[] symCompacte() {
// On vérifie d'abord si la matrice est symétrique
if (!estSymetrique()) {
```

```
return null; // Si ce n'est pas symétrique, on ne crée pas de tableau compact

}

// Calcul de la taille du tableau compact (n(n+1)/2)

int taille = n * (n + 1) / 2;

double[] c = new double[taille]; // Allocation du tableau compact

int k = 0; // Indice pour avancer dans le tableau compact c[]

// Parcours uniquement de la partie triangulaire supérieure

for (int i = 0; i < n; i++) {

for (int j = i; j < n; j++) {

c[k] = mat[i][j]; // Stocker M[i][j] dans c[]

k++;

}

}

return c; // Retourne le tableau compact

}

// ======================

// QUESTION 5 : Fonction acces() pour retrouver M[i][j] à partir du tableau compact

// ======================

public static double acces(double[] c, int n, int i, int j) {

// Grâce à la symétrie : si i > j on échange i et j

if (i > j) {

int tmp = i;

i = j;

j = tmp;

}
```

```
// Calcul de l'indice dans le tableau compact c[]
// Formule : index = i*(i+1)/2 + (j - i)
int index = (i * (i + 1)) / 2 + (j - i);
return c[index]; // On retourne la valeur stockée
}
// ======================
// QUESTION 6 : Fonction traiterLigne() pour parcourir la ième ligne
// ======================
public static void traiterLigne(double[] c, int n, int i) {
for (int j = 0; j < n; j++) {
double val = acces(c, n, i, j); // On récupère M[i][j] depuis c[]
System.out.print(val + " "); // Ici, traiterCoef(x) c'est afficher x
}
System.out.println(); // Retour à la ligne après avoir affiché la ligne i
}
// ======================
// QUESTION 7 : Fonction afficher() pour afficher toute la matrice
// ======================
public static void afficher(double[] c, int n) {
for (int i = 0; i < n; i++) {
traiterLigne(c, n, i); // On affiche chaque ligne
}
}
}
```

EXERCICE 7 :

1) Structure d'un arbre binaire d'entiers

```java
class ARB_BIN {
    int valeur; // étiquette (valeur) du nœud
    ARB_BIN gauche; // sous-arbre gauche
    ARB_BIN droite; // sous-arbre droit

    // Constructeur
    public ARB_BIN(int valeur) {
        this.valeur = valeur;
        this.gauche = null;
        this.droite = null;
    }
```

Cette classe représente un nœud d'arbre binaire qui contient une valeur entière et des références vers ses sous-arbres gauche et droit.

2) Procédure FEUILLE(A)

Une feuille est un nœud qui n'a pas d'enfants (ni gauche, ni droit). Pour afficher toutes les feuilles, nous utilisons un parcours en profondeur (DFS) et nous affichons un nœud quand il s'agit d'une feuille:

```java
public static void FEUILLE(ARB_BIN A) {
    if (A == null) {
        return;
    }

    // Si c'est une feuille (pas d'enfant gauche ni droit)
    if (A.gauche == null && A.droite == null) {
        System.out.print(A.valeur + " ");
    }

    // On visite toujours le côté gauche d'abord
    FEUILLE(A.gauche);
    FEUILLE(A.droite);
}
```

3) Procédure DEGRE(A)

Le degré d'un nœud est son nombre de fils. Pour un arbre binaire, le degré peut être 0, 1 ou 2:

```java
public static void DEGRE(ARB_BIN A) {
    if (A == null) {
        return;
    }

    // Calcul du degré
    int degre = 0;
    if (A.gauche != null)
        degre++;
    if (A.droite != null)
        degre++;

    // Affichage du nœud et son degré
    System.out.println("Nœud " + A.valeur + " : degré " + degre);

    // Récursion sur les sous-arbres
    DEGRE(A.gauche);
    DEGRE(A.droite);
}
```

4) Procédure pour trouver la profondeur d'un nœud

La profondeur d'un nœud est la longueur du chemin depuis la racine jusqu'à ce nœud. Pour trouver la profondeur d'un nœud contenant une valeur x:

```java
public static int profondeur(ARB_BIN A, int x) {
    return chercherProfondeur(A, x, niveau:0);
}

private static int chercherProfondeur(ARB_BIN A, int x, int niveau) {
    if (A == null) {
        return -1; // Nœud non trouvé
    }

    if (A.valeur == x) {
        return niveau; // Nœud trouvé à ce niveau
    }

    // Chercher à gauche
    int profondeurGauche = chercherProfondeur(A.gauche, x, niveau + 1);
    if (profondeurGauche != -1) {
        return profondeurGauche;
    }

    // Chercher à droite
    return chercherProfondeur(A.droite, x, niveau + 1);
}

// Pour afficher la profondeur
public static void afficherProfondeur(ARB_BIN A, int x) {
    int p = profondeur(A, x);
    if (p != -1) {
        System.out.println("La profondeur du nœud " + x + " est : " + p);
    } else {
        System.out.println("Le nœud " + x + " n'existe pas dans l'arbre.");
    }
}
```

5) Procédure HAUTEUR(A)

La hauteur d'un arbre est la longueur du plus long chemin de la racine à une feuille:

```java
public static int HAUTEUR(ARB_BIN A) {
    if (A == null) {
        return -1; // Hauteur d'un arbre vide est -1
    }

    // La hauteur est le max entre la hauteur du sous-arbre gauche et droit, plus 1
    int hauteurGauche = HAUTEUR(A.gauche);
    int hauteurDroite = HAUTEUR(A.droite);

    return Math.max(hauteurGauche, hauteurDroite) + 1;
}
```

6) Fonction SOM_NOEUDS(A)

Cette fonction calcule la somme de toutes les valeurs des nœuds dans l'arbre:

```java
public static int SOM_NOEUDS(ARB_BIN A) {
    if (A == null) {
        return 0;
    }

    // La somme est la valeur du nœud actuel plus les sommes des sous-arbres
    return A.valeur + SOM_NOEUDS(A.gauche) + SOM_NOEUDS(A.droite);
}
```

7) CLASSE MAIN

```java
public class Main {
    public static void main(String[] args) {
        // Création d'un arbre binaire de test
        ARB_BIN racine = new ARB_BIN(valeur:10);
        racine.gauche = new ARB_BIN(valeur:5);
        racine.droite = new ARB_BIN(valeur:15);
        racine.gauche.gauche = new ARB_BIN(valeur:3);
        racine.gauche.droite = new ARB_BIN(valeur:7);
        racine.droite.droite = new ARB_BIN(valeur:20);

        System.out.println(x:"Feuilles de l'arbre:");
        FEUILLE(racine);

        System.out.println(x:"\n\nDegrés des nœuds:");
        DEGRE(racine);

        System.out.println(x:"\nProfondeur du nœud 7:");
        afficherProfondeur(racine, x:7);

        System.out.println("\nHauteur de l'arbre: " + HAUTEUR(racine));

        System.out.println("\nSomme des nœuds: " + SOM_NOEUDS(racine));
    }
}
```

EXERCICE 8 :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// ---------------- STRUCTURES ----------------

typedef struct maillon {
        int numero;
        struct maillon *suiv, *prec;
} MAILLON, *PTR;

typedef struct lcb {
        PTR tete, queue;
} LCB;

typedef struct noeud {
        char *nom;
        LCB pages;
        struct noeud *gauche, *droit;
} NOEUD, *ABR;

// ---------------- FONCTIONS ----------------

LCB ajout_numero(int num, LCB numeros);
ABR ajout_nompropre(char* nom, int t[], int nombre, ABR a);
ABR supprimer_numero(char *nom, int numero, ABR a);
void afficher_index(ABR a);
void afficher_pages(LCB pages);

// ---------------- MAIN POUR TEST ----------------

int main() {
        ABR index = NULL;

        int fatou[] = {110, 250, 300};
        int mamadou[] = {3, 14, 101};
        int ousseynou[] = {11, 50};
        int pierre[] = {3, 7, 100, 287};
        int soda[] = {6, 10, 34, 66, 98};

        index = ajout_nompropre("Fatou", fatou, 3, index);
        index = ajout_nompropre("Mamadou", mamadou, 3, index);
```

```
        index = ajout_nompropre("Ousseynou", ousseynou, 2, index);
        index = ajout_nompropre("Pierre", pierre, 4, index);
        index = ajout_nompropre("Soda", soda, 5, index);

        printf("Index initial :\n");
        afficher_index(index);

        printf("\nSuppression de 250 pour Fatou :\n");
        index = supprimer_numero("Fatou", 250, index);
        afficher_index(index);

        return 0;
}

// ---------------- DEFINITIONS ----------------

LCB ajout_numero(int num, LCB numeros) {
        PTR nouveau = malloc(sizeof(MAILLON));
        nouveau->numero = num;
        nouveau->suiv = nouveau->prec = NULL;

        if (numeros.tete == NULL) {
        numeros.tete = numeros.queue = nouveau;
        return numeros;
        }

        PTR courant = numeros.tete;
        while (courant && courant->numero < num)
        courant = courant->suiv;

        if (courant && courant->numero == num) {
        free(nouveau);
        return numeros;
        }

        if (courant == numeros.tete) {
        nouveau->suiv = numeros.tete;
        numeros.tete->prec = nouveau;
        numeros.tete = nouveau;
        } else if (courant == NULL) {
        nouveau->prec = numeros.queue;
        numeros.queue->suiv = nouveau;
        numeros.queue = nouveau;
        } else {
        nouveau->suiv = courant;
```

```
        nouveau->prec = courant->prec;
        courant->prec->suiv = nouveau;
        courant->prec = nouveau;
        }

        return numeros;
}

ABR ajout_nompropre(char* nom, int t[], int nombre, ABR a) {
        if (a == NULL) {
        ABR nouveau = malloc(sizeof(NOEUD));
        nouveau->nom = strdup(nom);
        nouveau->gauche = nouveau->droit = NULL;
        nouveau->pages.tete = nouveau->pages.queue = NULL;
        for (int i = 0; i < nombre; i++)
        nouveau->pages = ajout_numero(t[i], nouveau->pages);
        return nouveau;
        }

        int cmp = strcmp(nom, a->nom);
        if (cmp == 0) {
        for (int i = 0; i < nombre; i++)
        a->pages = ajout_numero(t[i], a->pages);
        } else if (cmp < 0) {
        a->gauche = ajout_nompropre(nom, t, nombre, a->gauche);
        } else {
        a->droit = ajout_nompropre(nom, t, nombre, a->droit);
        }

        return a;
}

ABR supprimer_numero(char *nom, int numero, ABR a) {
        if (a == NULL) return NULL;

        int cmp = strcmp(nom, a->nom);
        if (cmp < 0) {
        a->gauche = supprimer_numero(nom, numero, a->gauche);
        } else if (cmp > 0) {
        a->droit = supprimer_numero(nom, numero, a->droit);
        } else {
        PTR courant = a->pages.tete;
        while (courant && courant->numero != numero)
        courant = courant->suiv;
```

```
        if (!courant) return a;

        if (courant->prec) courant->prec->suiv = courant->suiv;
        else a->pages.tete = courant->suiv;

        if (courant->suiv) courant->suiv->prec = courant->prec;
        else a->pages.queue = courant->prec;

        free(courant);
        }

        return a;
}

void afficher_pages(LCB pages) {
        PTR p = pages.tete;
        while (p) {
        printf("%d", p->numero);
        if (p->suiv) printf(", ");
        p = p->suiv;
        }
        printf("\n");
}

void afficher_index(ABR a) {
        if (a == NULL) return;
        afficher_index(a->gauche);
        printf("%s\t", a->nom);
        afficher_pages(a->pages);
        afficher_index(a->droit);
}
```