Licence **Sciences et Technologies**	**ANNEE : 2008/2009** **SESSION D'AUTOMNE 2008** **ETAPE :** CSB3, MHT53 **UE :** INF 251 **Epreuve :** Algorithmes et Structures de Données Fondamentaux **Date :** 23 Décembre 2008 **Heure :** 8H 30 **Durée :** 3 H **Documents :** Tous documents interdits **Vous devez répondre directement sur le sujet qui comporte 10 pages. Insérez ensuite votre réponse dans une copie d'examen comportant tous les renseignements administratifs** **Epreuve de** M^{me} Delest.

Indiquez votre code **d'anonymat** : N° : .

La notation tiendra compte de la clarté de l'écriture des réponses.

Barème indicatif

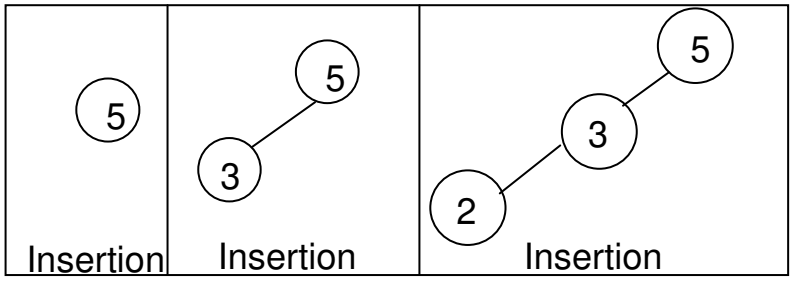
- Question 1 – Connaissances générales : 4 points
- Question 2 – Arbres binaires de recherche : 1.5 points
- Question 3 – B-Arbre : 1.5 points
- Question 4 – Utilisation des structures de données : 4 points
- Question 5 – Ecriture de fonction sur les arbres : 2 Points
- Question 6 – Ecriture de fonction sur les piles : 2 Points
- Question 7 – Listes et arbres binaires: 6 points

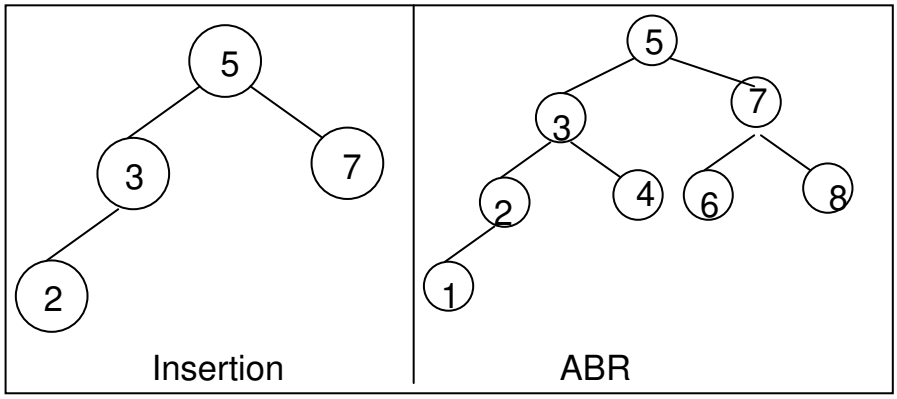
Question 1. Cochez les affirmations qui sont correctes :

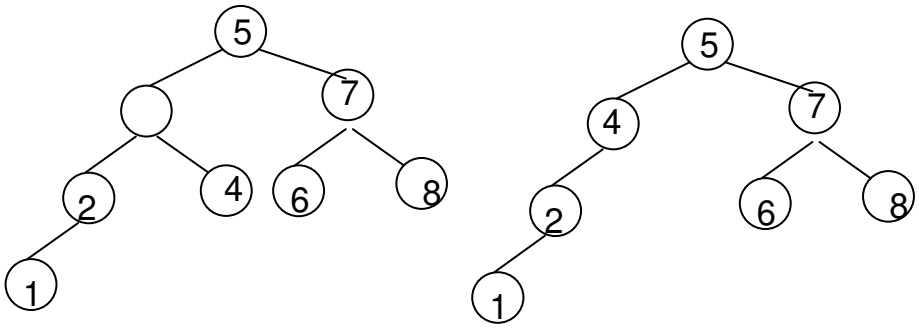
¤ Le temps d'accès au dernier élément d'une liste doublement chainée est en O(n).
¤ Dans un arbre binaire de recherche le minimum est toujours la feuille la plus à droite dans l'arbre.
x Le temps d'accès à l'élément maximum d'un tas max est en O(1).
x Un arbre AVL est un arbre binaire de recherche.
¤ Une table de hachage ouvert utilise le type pointeur.
x Dans un tas, la primitive supprimerValeur consiste à supprimer la valeur stockée à la racine de l'arbre.
x La structure de B-arbre est une généralisation des AVL

¤ Si s est une structure dont un des champs est n : ^entier, on accède à l'entier par s^.n

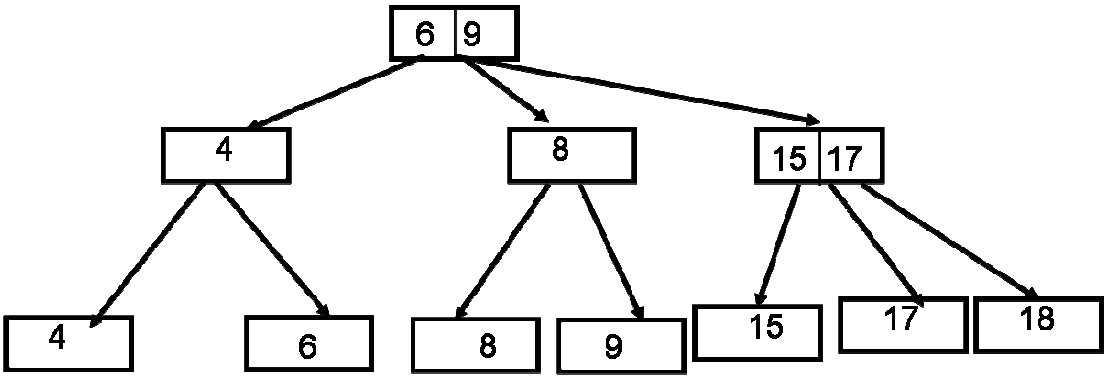
Question 2. Soit la suite de clé 5,3,2,7,6,8,4,1. Construire l'arbre binaire de recherche correspondant à l'insertion consécutive des clés, on dessinera l'arbre après chacune des quatre premières insertions ainsi que l'arbre final. Montrez l'exécution de la suppression de la clé 3 sur l'arbre binaire de recherche ainsi construit.



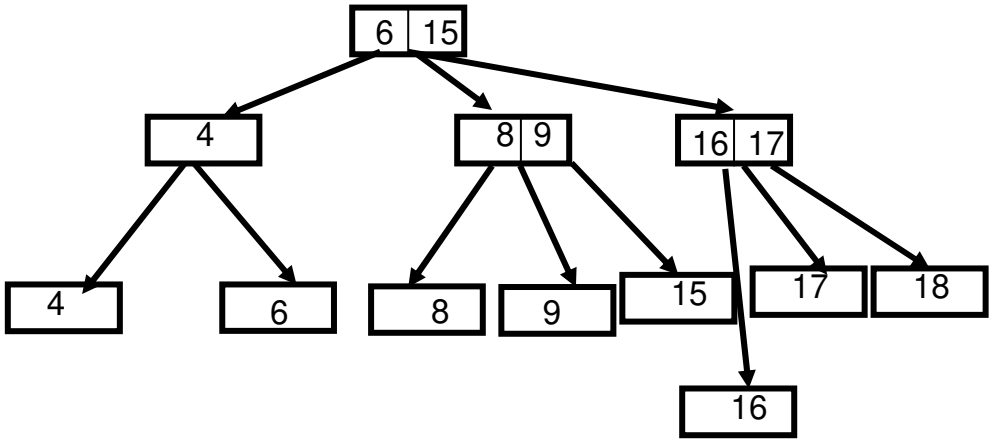




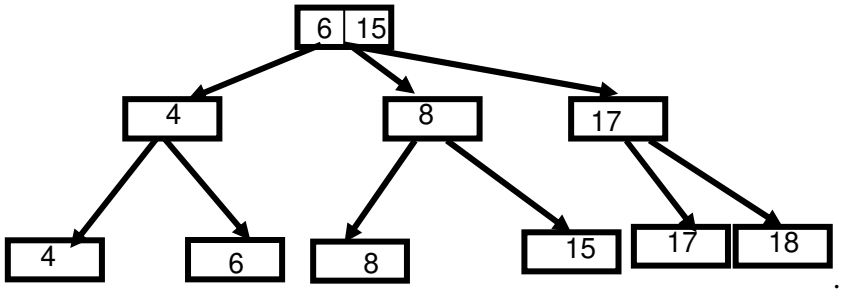
Question 3. On donne le 2-3-arbre suivant :



1 – Donnez sur l'arbre ci-dessus les positions des insertions qui conduiront à la création d'un trois nœud.
Toute adjonction dans l'arbre créera un trois nœud : par ajout sur les nœuds internes étiquetés 4 et 8, par ajustement du nœud interne étiqueté 15/17 qui donnera son fils le plus a gauche au nœud interne étiqueté 8.
2 - Donnez l'arbre après insertion de la valeur 16.



3 - Donnez l'arbre après suppression de la valeur 9 dans l'arbre de départ.



Question 4. Un concessionnaire de voitures d'occasion (spécialisé dans les voitures de moins de vingt an) souhaite enregistrer son stock avec les informations suivantes pour chaque voiture : marque (20 caractères), modèle (20 caractères), année, un certain nombre d'options présentes sur la voiture, chaque option codée avec un nombre de 0 à 30. **Attention, dans ce qui suit on ne s'intéresse qu'à quelques questions concernant cette gestion et pas à la totalité de la gestion d'une concession.**

1 – Décrire la structure de donnée voiture.

voiture=structure

marque : tableau[1..20] de caractères ;
modèle: tableau[1..20] de caractères ;
année :entier
listeOption :liste d'entier ;

finstructure

2 – Quelle structure de données peut-on utiliser pour un ensemble de voiture afin de répondre en temps constant à la question :

« voitures de marque x Année y »

où les valeurs x et y sont fixés au moment de la requête (par exemple x=Renault, y =2000)

Une table de hachage sur le champ marque permettra d'accéder directement aux voitures qui seront ainsi stockées. L'ensemble des voitures d'une même année peuvent être stockées dans une liste par année. Comme il y a peu d'années (voiture de moins de vingt ans) l'accès aux listes par année peut s'effectuer via un tableau.Donc le hachage de la clé donne accès à un tableau.

3 – Quelles structures de données peut-on utiliser pour accéder pour une option donnée à l'une des voitures les plus récentes ?

On peut utiliser un tas max et stocker chaque tas max dans un tableau indexé de 0 à 30 pour avoir un accès direct par option. L'ordre dans le tas max sera effectué par le champ année de la voiture.

4 – Définir le type abstrait concession qui permet de décrire un ensemble de voitures et de répondre aux questions 2 et 3 ainsi que ses primitives.

concession=structure

*T :table de hashage de marque ; /*adressage ouvert, il y a peu de marques*/*
/ on suppose la table de hashage de paramètre tailleTableHash */*
anMarque :tableau[0..tailleTableHash] de tableau[1988..2008] de liste de ^voiture
option :tableau[0..30] de tas min de ^voiture ;

finstructure
Une concession est un containeur.
créerConcession () :concession ;
détruireConcession (ref C :concession) :vide;
ajouterVoiture(ref C :concession ;ref c :^voiture) ;vide
supprimerVoiture(ref C :concession ;ref c :^voiture) ;vide ;
Des fonctions valeurs qui renverront des voitures. Par exemple :
trouverJeuneVoitureOption(ref C :concession ;ref x :entier) :^voiture ;
trouverAnnéeMarque (ref C :concession ;ref an :entier ;
ref m :tableau[1..20]de caractères) :liste de ^voiture ;
On aura besoin aussi des primitives
créerVoiture(ref v :voiture) :^voiture ;
détruireVoiture(ref v :^voiture) :vide ;

5 – Ecrire la fonction *ajouterVoiture* qui est appelée chaque fois qu'une nouvelle voiture doit être répertoriée dans la concession. On ne traitera pas les cas d'erreurs.

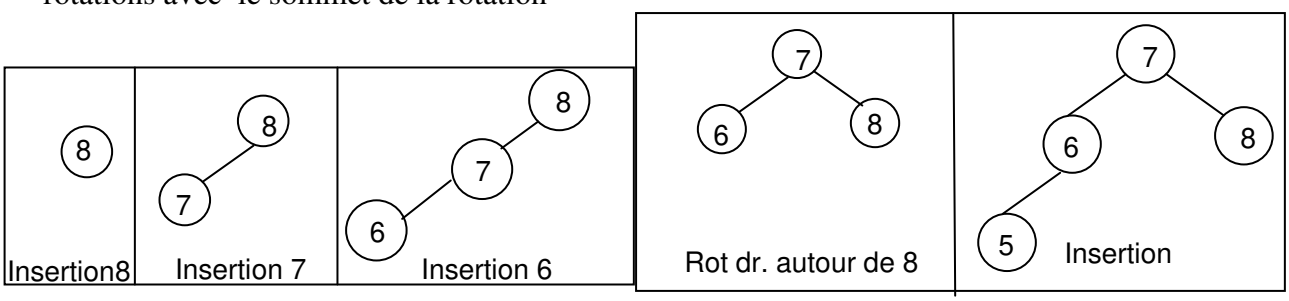
```
fonction ajouterVoiture(ref C :concession ;ref x :^voiture) :vide ;
        var p :curseur ;
        var b :booleen ;
        début
          debutListe(x^.listeOption) ;
          tant que !finListe(x^.listeOption) faire
                ajouterTasMin(C.option[valeur(x^.listeOption)],x)
          fintantque
         b= ajouter(T, x^.marque) ;
         p= chercher(T, x^.marque) ;
        ajouterEnTete(anMarque[p][x^annee],x) ;
        fin
```

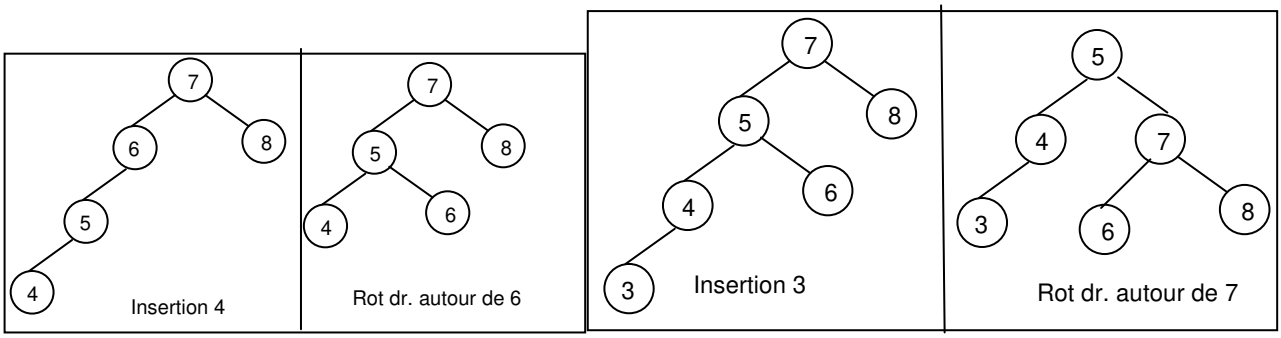
6 – Ecrire les deux fonctions correspondants aux questions 2 et 3.

```
fonction trouverAnnéeMarque (ref C :concession ;ref an :entier ;
                        ref m :tableau[1..20]de caractères) :liste de ^voiture ;
      var p :curseur ;
      début
          p= chercher(T, m) ;
          retourner anMarque[p][x^an]
      fin
fonction trouverJeuneVoitureOption(ref C :concession ;ref x :entier) :^voiture ;
      début
          retourner(valeur(option[x])
      fin
```

Question 5.

1 Dessiner l'arbre AVL correspondant à la liste de clé (8,7,6,5,4,3) . On donnera la liste des rotations avec le sommet de la rotation





2 - Ecrire une fonction qui teste si un arbre binaire de recherche est un arbre AVL dans chacun des cas suivants :

a. En utilisant des primitives d'arbre

Un arbre est AVL est un arbre binaire de recherche dont le facteur de balance est compris entre -1 et +1.

```
fonction estAVL(ref A :arbre binaire d'entier) :booleen ;
    retourner(estAVLbis(A,h)) ;
fonction estAVLBis(ref A :arbre binaire d'entier ;ref h :entier) :booleen ;
    var hg ;hd :entier ;
    var b :booléen ;
    début
        si estFeuille(A)  alors
            h=1 ;
            retourner(vrai)
        sinon
            hg=0 ;
            hd=0 ;
            si filGauche(A) !=NIL alors
                b= estAVLBis(filsGauche(A),hg) ;
            finsi
            si filsDroit(A) !=NIL et b alors
                b= estAVLBis(filsDroite(A),hd) ;
            finsi
            si b  et (abs(hd-hg)<2) alors
                h=max(hg,hd) ;
                retourner(vrai) ;
            sinon
                retourne(faux) ;
            finsi
        finsi
    fin
```

b. En utilisant les pointeurs.

```
fonction estAVLBis(ref A :arbre binaire d'entier ;ref h :entier) :booleen ;
    var hg ;hd :entier ;
    var b :booléen ;
    début
        si A^.gauche=NIL et A^.droit=NIL alors
            h=1 ;
            retourner(vrai)
        sinon
            hg=0 ;
            hd=0 ;
            b=vrai ;
            si A^.gauche !=NIL alors
                b= estAVLBis(A^.gauche,hg) ;
            finsi
```

```
            si A^.droit !=NIL et b alors
                b= estAVLBis(A^.droit !=NIL,hd) ;
            finsi
            si b  et (abs(hd-hg)<2) alors
                h=max(hg,hd) ;
                retourner(vrai) ;
            sinon
                retourne(faux) ;
            finsi
        finsi
    fin
```

Question 6. Ecrire une fonction qui prend en entrée une pile d'entier et qui modifie la pile en supprimant les entiers pairs et fournit en sortie la liste des entiers pairs supprimés. Par exemple, la pile [1,4,5,2,3,6,8] ou 1 est le fond de pile est transformée en la pile [1,5,3] ou 1 est le fond de pile et la liste fournie est (4,2,6,8). Donnez la complexité de votre fonction.

```
fonction pileListe(ref P :pile d'entier) :liste d'entier ;
    var L :liste d'entier ;
    var Ptmp :pile d'entier ;
    var v :entier ;
    début
        creerliste(L) ;
        creerPile(Ptmp) ;
        tantque !pileVide(P) faire
            v=valeur(P) ;
            si pair(v) alors
                insererEnTete (L,v)
            sinon
                empiler(Ptmp,v)
            finsi
            depiler(Ptmp)
        fintantque
        P=Ptmp ;
        detruire(Ptmp) ;
        retourner(L)
    fin
complexité O(taillePile)
```

Question 7. Lire l'ensemble du texte de la question AVANT de répondre. On pourra utiliser les fonctions vues en cours et en TD. On considère une liste de mots au sens de la définition du cours. La structure définie est la suivante :

```
mot=structure
        M :tableau[1..100] de caractère ; /* les mots sont limités à la longueur 100 */
        Long :entier ;/* sa longueur*/
finstructure
tout mot se termine par « \0 », la lettre « \0 » est plus petite que toute autre lettre.
```

1 - Donnez une liste de primitives pour le type *mot*. On précisera leurs fonctions.

Bien que de complexité non égale à O(1), la fonction concatener fait partie de primitives car c'est l'opération de base sur les mots. De plus, les mots sont de longueur connue au maximum 100 qui est une constante donc concatener restera en ce sens en temps constant.

```
fonction creerMot() :mot ;/* créé la liste et initialise la longueur*/
fonction  estVide(ref m :mot) :booléen ;/* teste si un mot est vide /*
fonction détruireMot(ref m :mot) :vide ;/* détruit la liste et initialise la longueur à 0*/
fonction getLongueur(ref m :mot) :entier ; /*longueur du mot en inclaunt « \ » */
fonction concatener(ref m,k :mot) :mot ; /* construit le mot mk */
fonction getLettre(ref m :mot ;val i :entier) :caractère ; /* ième lettre du mot */
```

fonction setLettre(ref m :mot ;val i :entier ;val c :caractère) :vide/ change la ième lettre */*

2 – Implémenter la fonction concatener qui colle deux mots l'un à la suite de l'autre et tronque le second mot si le tableau est saturé. Quelle est sa complexité ?

```
fonction concatener(ref m,k :mot) :mot ;
/* c'est une primitive on utilise la structure */
    var p :mot ;
    var i :entier ;
    début
        p=creerMot();
        p=m ;
        p.long=min(m.long+k.long-1,100) ;
        i=1;
        j=m.long
        tantque i<=k.longueur-1 et j<100 faire
            p.M[j]=k.M[i];
            i=i+1;
            j=j+1
        fintantque
        p.M[j]="\0";
        retourner(p) ;
    fin
```

complexité O(m.long+k.long) et comme la longueur des mots est bornée par 100, on peut considérer la cette fonction en O(1).

3 – Ecrire un opérateur *op::<* qui compare deux mots M1 et M2 et renvoit *vrai* si M1 est avant M2 dans l'ordre lexicographique (on écrira dans les algorithmes M1<M2).Quelle est sa complexité ?

```
fonction op ::<(ref m,k :mot) :booléen ;
/* ce n'est pas une primitive on n'utilise pas la structure */
    var mi,i :entier ;
    début
        i=1 ;
        tantque  getLettre(m , i)=getLettre(k, i) et getLettre(m,i) != « \0 » faire
            i=i+1
        fintantque
        retourner(getLettre(m,i)<getLettre(k,i))
    fin
```

complexité O(min(m.long,k.long)) et comme la longueur des mots est bornée par 100, on peut considérer la cette fonction en O(1).

4 – Ecrire une fonction *listeToDico* qui tranforme une liste de mots en dictionnaire. Etudiez sa complexité dans le pire des cas?

```
fonction listeToDico (ref L :liste de mot) :dico ;
    var D :dico ;
    début
        debutListe(L) ;
        creerDictionnaire(D) ;
        tantque !finListe(L) faire
            si !appartient(D,valeur(L)) alors
                ajouter(D,valeur(L))
            finsi
            suivant(L)
         fintantque
        retourner(D)
    fin
```

Soit n la taille de la liste. Dans le pire des cas toutes les lettres sont différentes et dans un sommet de l'arbre. Il va falloir créer autant de branche gauche que de mots. Cette opération pour chaque mot s'effectue en O(1) puisque la longueur des mots est bornée par 100 donc en O(n) pour la liste. Il faudra insérer ces branches gauches dans la branche droite du dictionnaire. Si tout se passe mal, les mots sont dans l'ordre alphabétique dans la liste ce qui fait que pour chaque mot on doit balayer la branche droite qui s'agrandit à chaque adjonction. On aura une complexité en O(n^2) pour cette insertion (branche de longueur 1 puis 2 puis 3 …. puis n-1 donc 1+2+3+…+n-1.). Si on garde le pointeur du dictionnaire sur le dernier on n'améliore pas la complexité car c'est alors l'ordre lexicographique inverse qui sera le cas le pire. Par suite, on peut considérer que cette fonction est en O(n^3).

5 – Ecrire une fonction *corrige* qui modifie la dernière lettre d'un mot du dictionnaire. On supposera que le mot se trouve dans le dictionnaire. Cette fonction prendra en argument un dictionnaire, le mot correct ainsi que la lettre à changer.
On utilise les deux primitives ajouter et supprimer ! Bien sûr, on peut corriger la dernière lettre et c'est plus efficace puisque on n'utilisera pas de fonction new et delete. Cependant, c'est plus délicat avec quatre cas de figures déterminés par

- *il y a un fils droit sur de « \0 »*
- *le père de « \0 » est un fils gauche ou un fils droit*

```
fonction corrige (ref D :dico ;ref k :mot ;val c :caractère) :vide ;
    var ctmp :caractère ;;
    var i :entier ;
    début
        i= getLongueur(k)-1 ;
        ctmp=getLettre(k,i) ;
        setLettre(k,i,c) ;
        supprimer(D,k) ;
        setLettre(k,i, ctmp) ;
        ajouter(D,k) ;
    fin
```

6 – On souhaite stocker des informations associées aux mots du dictionnaire de façon à y accéder rapidement. Une information est composée d'une phrase ainsi que d'un certain nombre de mots associés appartenant au dictionnaire. Expliquez comment modifier la structure du dictionnaire.
Une information peut être stockée via un pointeur tout comme un sommet. Une opération « cast » (vue en programmation) permet de transformer un pointeur vers un type x en un pointeur vers un type y.
Par ailleurs, le sommet contenant la valeur « \0 » n'aura pas de fils gauche car par définition c'est la fin du mot. On peut donc utiliser le pointeur vers le fils gauche pour pointer l'information.

Question bonus : Définir le type abstrait *informationPourDico* et donner la nouvelle définition de la structure de donnée dico. Donnez la liste des primitives.

```
informationPourDico=^ information
information = structure
                    phrase=mot ;
                    motAssocies=liste de mot ;
              finstructure
```

La structure dico reste inchangée ainsi que ses primitives de départ. On ajoute uniquement des primitives de manipulation de l'information.

```
fonction creerInformationPourDico( ref ph :mot ; ref motA: liste de mot) : informationPourDico
fonction detruireInformationPourDico( ref p: informationPourDico) :vide ;
fonction ajouterInformationDico(ref D :dico,ref M :mot ;ref inf : informationPourDico) ;
fonction supprimerInformationDico(ref D :dico ;ref M :mot) : informationPourDico ;
```

On peut agrémenter par des primitives de manipulation des mots associés :

```
fonction ajouterMotAssocié (ref D :dico,ref M :mot ;ref ma :mot) :vide ;
fonction supprimerMotAssocié(ref D :dico ;ref M :mot ;ref ma :mot) : booléen;
```

Listes simplement chainées (listeSC)

fonction valeur(val L:liste d'objet):objet;
fonction debutListe(val L:liste d'objet);
fonction suivant(val L:liste d'objet);
fonction listeVide(val L:liste d'objet): booleen;
fonction créerListe(ref L:liste d'objet):vide;
fonction insérerAprès(ref L:liste d'objet; val x:objet;):vide;
fonction insérerEnTete(ref L:liste d'objet val x:objet):vide;
fonction supprimerAprès(ref L:liste d'objet):vide;
fonction supprimerEnTete(ref L:liste d'objet):vide;

Listes doublement chainées (listeDC)

fonction finListe(val L:liste d'objet):vide;
fonction précédent(val L::liste d'objet): vide;

Piles

fonction valeur(ref P:pile de objet):objet;
fonction fileVide(ref P:pile de objet):booléen;
fonction créerPile(P:pile de objet) :vide
fonction empiler(ref P:pile de objet;val x:objet):vide;
fonction dépiler(ref P:pile de objet):vide;
fonction detruirePile(ref P:pile de objet):vide;

Files

fonction valeur(ref F:file de objet):objet;
fonction fileVide(ref F:file de objet):booléen;
fonction créerFile(F:file de objet);vide;
fonction enfiler(ref F:file de objet;val x:objet):vide;
fonction défiler (ref F:file de objet):vide;
fonction detruireFile(ref F:file de objet):vide;

Arbres binaires

fonction getValeur(val S:sommet):objet;
fonction filsGauche(val S:sommet):sommet;
fonction filsDroit(val S:sommet):sommet;
fonction pere(val S:sommet):sommet;
fonction setValeur(ref S:sommet;val x:objet):vide;
fonction ajouterFilsGauche(ref S:sommet,val x:objet):vide;
fonction ajouterFilsDroit(ref S:sommet,x:objet):vide;
fonction supprimerFilsGauche(ref S:sommet):vide;
fonction supprimerFilsDroit(ref S:sommet):vide;
fonction detruireSommet(ref S:sommet):vide;
fonction créerArbreBinaire(val Racine:objet):sommet;

Arbres planaires

fonction valeur(val S:sommetArbrePlanaire):objet;
fonction premierFils(val S:sommetArbrePlanaire):sommetArbrePlanaire;
fonction frere(val S:sommetArbrePlanaire):sommetArbrePlanaire;
fonction pere(val S:sommetArbrePlanaire):sommetArbrePlanaire;
fonction créerArborescence(val racine:objet):sommetArbrePlanaire;
fonction ajouterFils(ref S:sommetArbrePlanaire,val x:objet):vide;
fonction supprimerSommet(ref S:sommetArbrePlanaire):vide;
fonction créerArbreBPlaniare(val Racine:objet):sommet;

Arbres binaire de recherche

fonction ajouter(ref A:arbreBinaire d’objets, val v:objet):vide;
fonction supprimer(val A: arbreBinaire d’objets, val v:objet):vide;

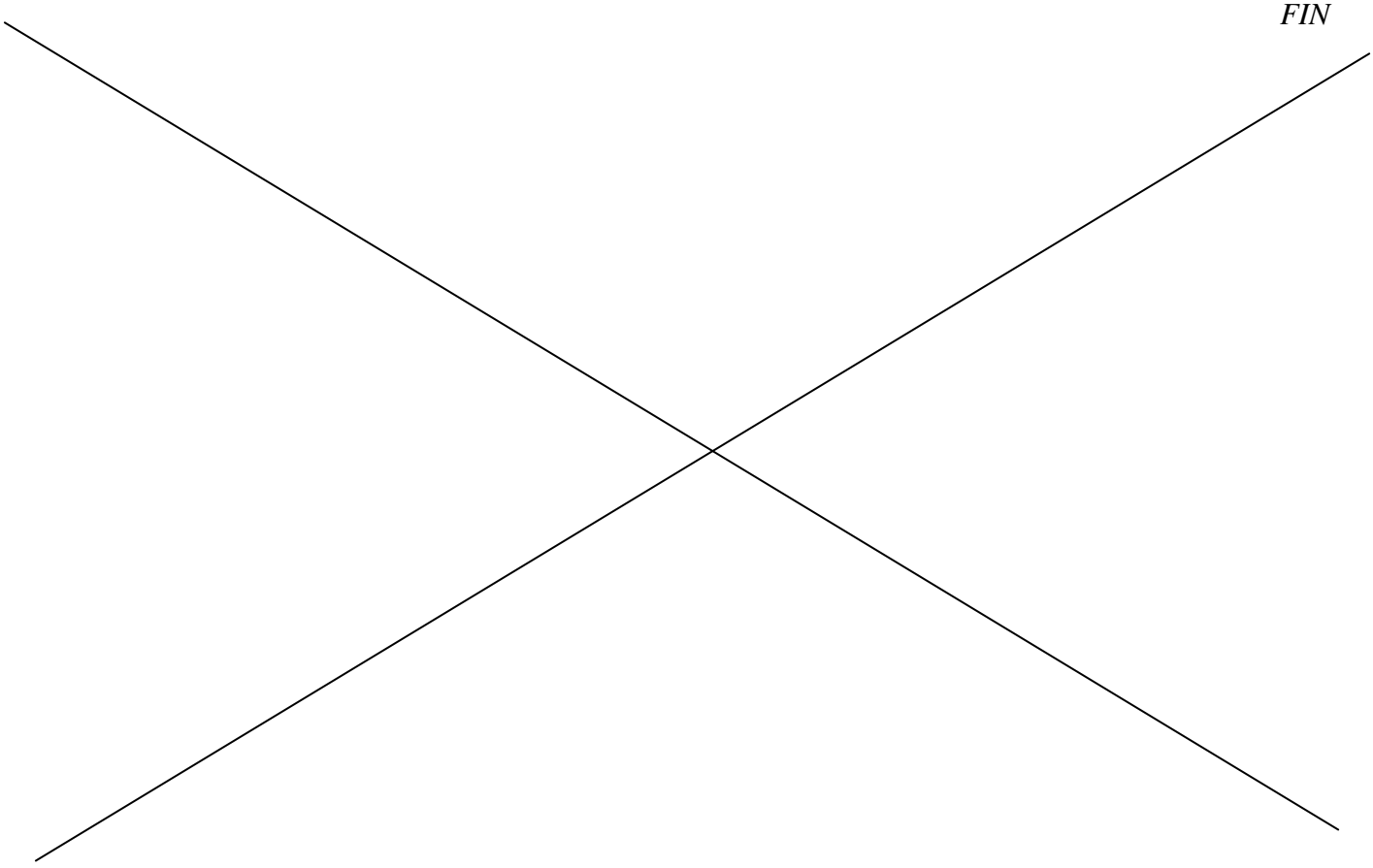
Tas

fonction valeur(ref T:tas d'objet): objet;
fonction ajouter(ref T:tas de objet, val v:objet):vide;
fonction supprimer(val T:tas de objet):vide;
fonction creerTas(ref T:tas,val:v:objet):vide;
fonction detruireTas(ref T:tas):vide;

Dictionnaire

fonction appartient((ref d:dictionnaire,val M::mot):booléen;
fonction creerDictionnaire(ref d: dictionnaire):vide ;
fonction ajouter(ref d:dictionnaire,val M::mot):vide;
fonction supprimer(ref d:dictionnaire,val M:mot):vide;
fonction detruireDictionnaire(ref d:dictionnaire):vide;

Table de Hachage

fonction chercher(ref T:tableHash de clés, val v:clé):curseur;
fonction créerTableHachage(ref T: tableHash de clé, ref h:fonction):vide;
fonction ajouter(ref T:tableHash de clé,val x:clé):booleen;
fonction supprimer((ref T:tableHash de clé,val x:clé):vide;

FIN