Licence **Sciences et Technologies**	**ANNEE : 2008/2009** **SESSION DE PRINTEMPS 2009** **ETAPE :** CSB3, MHT53 **UE :** INF 251 **Epreuve :** Algorithmes et Structures de Données Fondamentaux **Date :** 8 Juin 2009 **Heure :** 14H **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, files, listes : 2 Points
- Question 7 – Listes et tas : 5 points

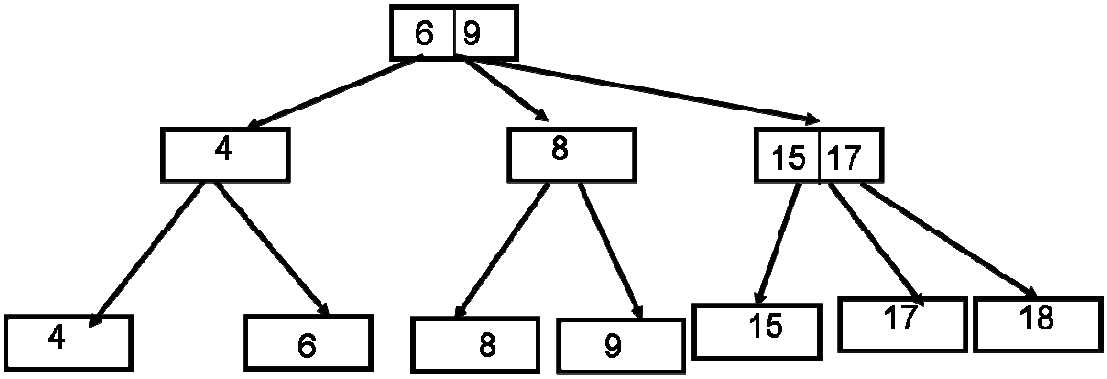
Question 1. Cochez les affirmations qui sont correctes :

☐ Le temps d'accès au dernier élément d'une liste simplement chainée est en O(1).
☐ Dans un arbre binaire de recherche le minimum est toujours la feuille la plus à gauche dans l'arbre.
☐ Le temps d'accès à l'élément maximum d'un tas min est en O(1).
☐ Un arbre AVL peut parfois être un tas..
☐ Une table de hachage à adressage chainé utilise le type pointeur.
☐ Dans un tas, la primitive ajouterValeur consiste à ajouter la valeur stockée à la racine de l'arbre.
☐ La structure de B-arbre permet de diminuer le temps d'accès à un élément.

☐ Si s est un pointeur vers 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 le tas max 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 6 sur le tas max ainsi construit.

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



Chaque question est à traiter sur l'arbre ci-dessus.

1 – Donnez sur l'arbre ci-dessus les feuilles s tels que leur suppression conduira à une suppression du nœud père de s.

2 - Donnez l'arbre après insertion de la valeur 17.5.

3 - Dessinez et expliquez les modifications de l'arbre lors de la suppression de la valeur 8.

Question 4. On désire modéliser le comportement d'un client dans un cinéma de son entrée à la sortie du cinéma. Un client sera modélisé comme un objet de type abstrait *client* que l'on ne décrit pas ici. On suppose que le cinéma a une seule porte d'entrée, trois guichets et 24 salles de cinéma, chacune ayant une sortie. On accède aux salles 1 à 12 par un point commun de contrôle des tickets. Il en est de même pour les salles 13 à 24. Pour simplifier, on supposera toutes les salles identiques et que chaque rangée de la salle a le même nombre de siège. **Attention, dans ce qui suit on ne s'intéresse qu'à quelques questions concernant cette modélisation et pas à la totalité.**

1. Quelles structures de données peut-on utiliser pour l'accès d'un client au cinéma ? aux guichets ? aux points de contrôle ? à la porte de sortie d'une salle ?
2. Quelle structure de données peut-on utiliser pour la salle de cinéma ?
3. Définir le type abstrait *cinéma* support à cette modélisation. Donnez les deux primitives obligatoires.
4. Donner trois exemples de primitives autres que celles de la question précédente.
5. Ecrire la fonction *vaGuichet* intervenant dans cette modélisation qui donne accès d'un client au guichet.
6. Ecrire la fonction *urgence* qui vide les salles de cinéma. On considèrera que les clients en attente au guichet et points de contrôle sont évacués par une autre fonction.

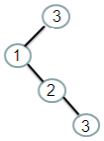
Question 5.

1 Dessiner l'arbre AVL correspondant à la liste de clé (3,8,7,6,4,2). On donnera la liste des rotations avec le sommet de la rotation

2 Ecrire une fonction *deuxPlusPetit* qui donne le second plus petit élément d'un arbre AVL (dans l'exemple la valeur 3)

- En utilisant des primitives d'arbre
- En utilisant les pointeurs.

Question 6. Ecrire une fonction *nonTrois* qui prend en entrée une file d'entiers non nuls et qui exploite la file en mettant dans une pile les entiers non multiples de 3 et dont le résultat est un arbre binaire de recherche des entiers supprimés divisés par trois. La file est inchangée après exécution de la fonction. Par exemple, la file [9,1,4,5,2,3,6,8,9[donne la pile [1,4,5,2,8 et l'arbre fourni est



Donnez et expliquez la complexité de votre fonction. Ecrire une fonction *compte* qui prend en entrée un arbre binaire de recherche et compte le nombre d'entiers impairs fils gauche dans l'arbre binaire de recherche. Ce calcul pouvait-il être fait dans la fonction *nonTrois* ? Si non, pourquoi ? Si oui, comment ?

Question 7. Lire l'ensemble du texte de la question AVANT de répondre. On pourra utiliser les fonctions vues en cours et en TD en précisant l'en-tête de la fonction et son fonctionnement. On considère une forme planaire dont le contour est donné par une liste de points du plan $\mathbb{R} \times \mathbb{R}$. Par exemple, la liste F=[(0,0),(1,0),(1,1),(0,1)] donne le contour d'un carré.

1 - Donnez une définition du type *point*.

2 - En précisant leur rôle, donnez une liste de primitives pour le type *point*. On précisera l'en-tête de ces primitives. On ne demande pas le code des primitives.

3 - Le type abstrait *forme* est une liste de points. Est-ce une liste simplement chainé ou doublement chainée ? Justifiez.

4 - On appelle taille de la forme le nombre de points d'une forme (dans l'exemple *taille*(F)=4). Ecrire la fonction *taille*. Quelle est sa complexité ?

5 - Ecrire une fonction *intersection* qui trouve une liste des points communs à deux formes. Etudiez sa complexité.

6 - Ecrire une fonction *casser* qui insère entre deux points z1 et z2 le point milieu du segment [z1, z2].

7 - On associe à une forme sa hauteur par rapport au plan de base. Les formes peuvent ainsi être empilées sur un écran à la manière des fenêtres de travail (une forme de hauteur 1 est au-dessous). Décrire la structure de donnée *formeEmpile* correspondant qui prend en compte cet ajout.

8 - On désire implémenter deux fonctions de manipulation des formes

- Passer une forme au dernier plan (fonction *dPlan*)
- Passer une forme au premier plan (fonction *pPlan*)

Expliquez pourquoi la structure de tas est adaptée.

9 - Donner la structure de donnée *tasFormeEmpile*. Ecrire la fonction *dPlan*.

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;

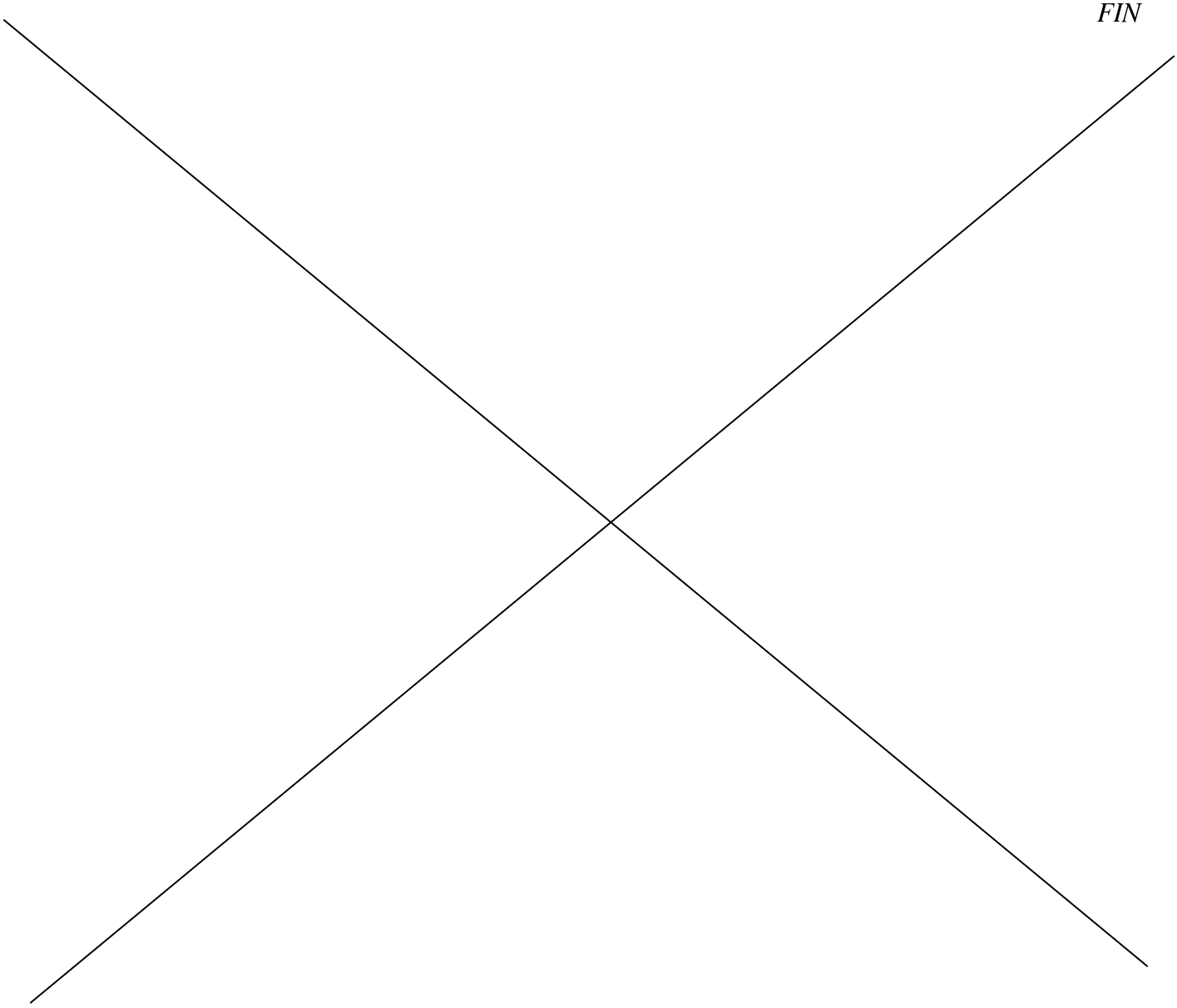
File de priorité

fonction changeValeur(ref T:tas d'objet,val s:sommet,val v:objet):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