DEVUIP Service Scolarité	**ANNEE UNIVERSITAIRE 2011/2012** **SESSION DE PRIMPTEMPS** PARCOURS / ETAPE : LIIN300 Code UE : J1IN3001 **Epreuve :** Algorithmique et structures de données 1 **Date :** **Heure :** **Durée :** 1H30 Documents : non autorisés Epreuve de Mme : DELEST **Vous devez répondre directement sur le sujet qui comporte 8 pages. Insérez ensuite votre réponse dans une copie d'examen comportant tous les renseignements administratifs**	

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

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

Barème indicatif

- Question 1 – Connaissances générales : 2 points
- Question 2 – Containeur liés aux arbres binaires : 6 points
- Question 3 – Connaissance des structures de données : 6 Points
- Question 4 – Utilisation des structures de données : 6 points

Question 1. Cochez les affirmations correctes et quelle que soit la réponse justifiez sur les lignes en pointillé.

☐ La complexité mémoire d'une table de hachage chaînée est plus importante que la complexité mémoire d'une table de hachage ouvert.

……
……

☐ Un tas est un arbre binaire de recherche.

……
……

☐ La structure de file permet l'accès au premier élément entré dans le containeur.

……
……

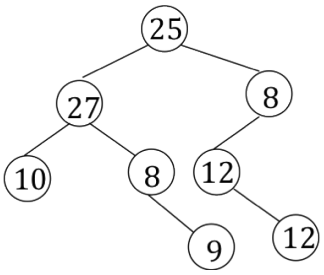
☐ Si s est un pointeur vers un entier, on modifie la valeur de l'entier en écrivant s=s+1

……
……

Question 2. Soit la liste de clé A=(22,5,12,2,16,7,1,23).

1. Construire l'arbre binaire de recherche T correspondant à l'insertion consécutive des clés de la liste A. On dessinera l'arbre après chacune des quatre premières insertions ainsi que l'arbre final.
2. Montrez l'exécution de la suppression du plus petit élément sur l'arbre T.
3. Ecrire la fonction *supprime* qui supprime l'élément le plus petit dans un arbre binaire de recherche.
4. Donner la liste des sommets en ordre préfixe, en ordre infixe et en ordre suffixe.
5. Construire le tas MAX B correspondant à l'insertion consécutive des clés de la liste A. On dessinera l'arbre après chacune des quatre premières insertions ainsi que l'arbre final.
6. Construire l'arbre binaire de recherche AVL correspondant à l'insertion consécutive de la liste A. On dessinera l'arbre après chacune des quatre premières insertions ainsi que l'arbre final.
7. Construire le 2-3 arbre correspondant à l'insertion consécutive de la liste A. On dessinera l'arbre après chacune des quatre premières insertions ainsi que l'arbre final.

Question 3. Soit un arbre binaire dont la valeur contenue dans les sommets est un entier positifs non nuls. On appellera un tel arbre N-arbre. La figure ci-dessous donne un exemple de N-arbre.



1. Ecrire la fonction *verifier* qui vérifie qu'un arbre binaire est un N-arbre.
2. En utilisant un parcours hiérarchique et les primitives du type abstrait listeSC, écrire la fonction *arbreListes* qui prend pour paramètre un tel arbre et fournit en sortie un tableau T tel que T[i] donne accès à la liste des valeurs des sommets de hauteur i dans l'arbre. On pourra utiliser la fonction *hauteurArbreBinaire* vue en cours sans la réécrire et dont l'en-tête est :

```
fonction hauteurArbreBinaire(val s:sommet):entier ;
```

Sur l'exemple, T[0]=[25], T[1]=[27,8], T[2]=[10,8,12],T[3]=[9,12].

1. Pour un sommet s, soit G(s) (resp. D(s)) l'ensemble des sommets du sous-arbre gauche (resp. droit) de s. On dit qu'un sommet s est sympathique si

$$valeur(s) = \sum_{s' \in G(s)} valeur(s') + \sum_{s' \in D(s)} valeur(s').$$

En d'autres termes, la valeur du sommet s est la somme des valeurs des sommets de ses sous-arbres. Ecrite la fonction *nbSympa* qui prend pour argument un N-arbre et fournit comme résultat le nombre de nœud sympathique de l'arbre. Sur l'exemple, le résultat est 2.

Question 4. Pour gérer une flotte de bateau de transport de passager et animer le point d'embarquement des passagers, une ville décide de mettre en place un système visuel permettant au point d'embarquement de connaître la position des bateaux sur la carte du lieu. Pour cette raison, chaque bateau peut transmettre une information comportant sa géolocalisation (une latitude et une longitude) ainsi que un signal correspondant à l'état de la mer : calme, agitée, très agitée, forte. La latitude et la longitude sont données par un triplet (degré, minute, seconde). Chaque bateau est immatriculé par une suite de 7 caractères. On s'intéresse uniquement à ce que ces informations soient accessibles en temps réel pour les usagers sans prendre en compte la cartographie, les pannes de bateau ou la possibilité d'embarquement des passagers. Il est plus important pour un usager de savoir que la mer est très forte que de savoir qu'elle est calme (il pourra renoncer à embarquer. **Attention, dans ce qui suit on ne s'intéresse qu'aux questions ci-dessous pas à la totalité du problème.**

Dans la suite, pour chaque structure de donnée, on précisera s'il s'agit d'un containeur, on décrira précisément la structure. On ne demande pas d'écrire les primitives mais uniquement de donner l'en-tête de la primitive et son rôle. Justifier vos choix !!!

1. Décrivez la structure de données *localisation* permettant de stocker les géolocalisations. En donner les primitives.
2. Décrivez la structure de données *bateau* permettant de décrire les informations liées à un bateau. En donner les primitives.
3. Décrivez la structure de données *tableBateau* permettant de mettre à jour efficacement les informations liées à un bateau donné. En donner les primitives.
4. L'application qui utilisera ces structures devra lorsque le bateau signale une mer forte sur un passage particulier agir immédiatement sur l'afficheur pour signaler fortement ce problème aux usagers. Proposer une structure de donnée permettant d'accéder facilement aux bateaux à signaler. Justifiez.
5. Ecrire la fonction *ajouterBateau* qui prend comme paramètre un bateau et permet d'insérer le bateau dans toutes les structures de données que vous venez de décrire. **Attention à utiliser les primitives vues en cours et données en annexe.**

Listes simplement chainées (listeSC)

fonction valeur(val L:liste d'objet):objet;
fonction debutListe(val L:liste d'objet) :vide ;
fonction suivant(val L:liste d'objet) :vide ;
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;
fonction detruireListe(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 pileVide(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;
fonction detruireArbreBinaire(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 detruireArborescence(val racine:objet):sommetArbrePlanaire;

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;
fonction detruireTableHachage(ref T: tableHash de clé, ref h:fonction):vide;

FIN