DS n° 2 Informatique III

Calculatrice et documents non autorisés

Lorsqu'aucune consigne n'est précisée, les réponses peuvent être données en langage C ou en pseudo-code.

Vous pouvez définir autant de fonctions que vous le souhaitez dans chaque exercice. Il n'y a pas nécessairement l'équivalence : une question = une fonction.

Le barême est donné à titre indicatif mais peut être sujet à changement.

Exercice 1 (*Gestion d'arbres ternaires*)(9 pts)

1. Déclarer une structure `Arbre` pour représenter un nœud d'arbre ternaire (au maximum **trois** fils) stockant un entier.
2. Écrire une fonction `initialiserNoeud` qui prend en paramètre un entier et retourne un pointeur vers un nouveau nœud initialisé avec la valeur passée en argument.
3. Écrire une fonction `compterEnfants` qui retourne le nombre d'enfants d'un nœud passé en paramètre (cette valeur pouvant être 0, 1, 2 ou 3).
4. Pour un arbre ternaire, un parcours préfixe fonctionne ainsi :
   — affichage de la racine
   — affichage du fils de gauche
   — affichage du fils du milieu
   — affichage du fils de droite
   Écrire une fonction/procédure `parcoursPrefixe` qui prend en paramètre un pointeur vers la racine de l'arbre et affiche simplement les valeurs des nœuds en utilisant un tel parcours préfixe.
5. Nous allons implémenter un parcours en largeur pour un tel type d'arbre :
   (a) Ecrire la/les structure(s) nécessaires pour implémenter ce parcours.
   (b) **On suppose que les fonctions `empiler` et `depiler` (pour une pile) ainsi que `enfiler` et `defiler` (pour une file) ont déjà été écrites.**
   Si ces fonctions doivent être utilisées, indiquer leurs prototypes pour qu'elles correspondent aux structures déclarées à la question précédente.
   (c) Ecrire la fonction `parcoursLargeur` qui prend en paramètre un pointeur vers la racine de l'arbre et affiche les valeurs des nœuds en utilisant un parcours en largeur.

Exercice 2 (*Fusion d'ABR*)(4 pts)

Dans cet exercice, on suppose que chaque noeud d'un ABR ne contient qu'un entier et les deux pointeurs vers les fils. On suppose aussi que la fonction/procédure de création de noeud existe également. Enfin on suppose que la fonction pour ajouter un entier dans un ABR existe également.

1. Définir la structure d'un noeud d'un arbre ABR pour bien préciser les noms des champs à utiliser.
2. Définir le prototype de la fonction/procédure de création de noeud pour bien préciser les paramètres et/ou la valeur de retour.
3. Définir le prototype de la fonction/procédure d'ajout d'un entier dans un ABR pour bien préciser les paramètres et/ou la valeur de retour.
4. Écrire une fonction/procédure `fusionABR`. Cette fonction/procédure va effectuer les tâches suivantes :
   — prendre deux Arbres Binaires de Recherche (ABR) en paramètre
   — créer un troisième arbre contenant toutes les valeurs des deux premiers (les éléments peuvent être ajoutés dans n'importe quel ordre tant que l'arbre final contient bien toutes les valeurs des deux arbres d'origine). Les doublons ne seront pas ajoutés dans l'arbre final.
   — retourner l'adresse de la racine du troisième arbre créé.

Exercice 3 (*Arbres et sécurité sociale*)(7 pts)

On souhaite stocker dans un système les données personnelles d'un grand groupe de personnes. Chaque usager est identifié par son numéro de sécurité sociale unique. Dans le but d'optimiser les recherches de ces personnes, nous utiliserons une structure de type ABR (Arbre de Recherche Binaire) qui prendra comme élément de référence ce numéro unique de chaque personne (l'ABR est donc rangé en fonction des numéros de sécurité sociale).

1. Justifier le choix d'utilisation de l'ABR, par rapport à d'autres types de structures, pour pouvoir stocker les informations des personnes.
2. Quel type de parcours d'arbre faudrait-il effectuer si on voulait afficher tous les numéros des usagers présents dans l'arbre par ordre décroissant ? Justifier.
3. Définir la structure `Personne` contenant au moins son nom, son prénom, son âge (nombre positif stocké sur pas plus de 3 chiffres en base 10), ainsi que son numéro de sécurité sociale (nombre positif toujours sur 15 chiffres en base 10).
4. Définir la structure `Arbre` qui contient une structure `Personne` allouée dynamiquement, ainsi que 2 pointeurs vers des structures `Arbre`. Définir également un type `Parbre` qui pointera vers un nœud de l'ABR, donc vers une structure `Arbre`.
5. Écrire une fonction `Parbre creation_noeud(Personne* p1)` qui permet de créer un nœud de l'arbre contenant les données d'une personne passée en paramètre. Cette fonction retourne un pointeur vers le nœud créé. Attention, cette fonction doit copier les données provenant de l'adresse en paramètre vers la structure allouée. Il faut partir du principe que la structure pointée par p1 va être libérée par la fonction appelante après l'appel à `creation_noeud`. En d'autres termes, une copie de la personne passée en paramètre doit être effectuée dans le noeud qui va être créé ici.
6. Écrire une fonction `Parbre insertion(Parbre arbreSecu, Personne* p1)` permettant d'ajouter à l'ABR `arbreSecu` un nouveau nœud contenant la personne passée en paramètre (attention, le nœud doit être correctement placé). Si cette personne ajoutée est déjà existante, alors le programme devra afficher un message d'erreur mais ne devra pas s'arrêter.
7. Écrire une fonction/procédure `recherche(...)` qui permet de rechercher une personne à partir d'un numéro de sécurité sociale, et de retourner l'ensemble de ses informations si elle est présente dans l'ABR, ou renvoyer une valeur particulière pour indiquer qu'elle n'existe pas sinon.
8. Écrire une procédure `void afficheSecu(Parbre arbreSecu)` qui va afficher les numéros de sécurité sociale des usagers présents dans l'arbre pointé par `arbreSecu` en faisant un parcours suffixe.