LO21-2011A-FSOA-01

Examen Final – LO21 et LO27
2 heures
Aucun document autorisé

Le barème par exercice est donné à titre indicatif et pourra être modifié durant la correction

Exercice 1: Gestion d'une salle de spectacle (10 points)

Remarque : Dans cet exercice, vous pouvez utiliser les fonctions définies en cours sur le type de donnée abstrait Liste (est_vide, insérer_tête, insérer_queue, etc.)

On considère une rangée dans une salle de spectacle comme une liste de places. Chaque place est caractérisée par son numéro ou son rang (un entier de 1 à N, avec N inconnu) et son état (libre ou occupée). Une rangée est triée par ordre croissant sur les numéros de places.

Question 1 : (2 points)
Étant donnée une rangée, écrire l'algorithme récursif du sous-programme **nbFreePlace** permettant de calculer le nombre de places libres.

Question 2 : (3 points)
Étant données 2 rangées, écrire l'algorithme récursif du sous-programme **commonFreePlace** permettant de calculer le nombre de places libres se trouvant dans le même rang dans les deux rangées.

On représente une salle de spectacle par une liste chaînée de rangées numérotées. Chaque élément de cette liste est caractérisé par le numéro de la rangée (un entier) et la rangée elle-même représentée par une liste chaînée de places. Une place est définie comme précédemment (numéro de la place et son état).

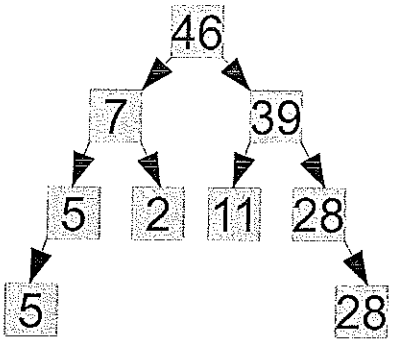
Question 3 : (2 points)
Donner en C la déclaration des types nécessaires à la déclaration d'une salle : le type *Place*, le type *Rangée*, le type *Salle*.

Question 4 : (3 points)
Étant donnée une salle, écrire en C le sous-programme itératif **getFreePlaces** permettant de construire une nouvelle liste *LibreListe* contenant toutes les places libres de la salle et leur numéro de rangée. Chaque élément de cette nouvelle liste est un couple composé du numéro de la rangée et du numéro de la place libre.
Donner en C la déclaration du type *LibreListe*.

Exercice 2 : Arbres binaires et Arbres binaires de recherche (7 points)

Question 1 : Arbre binaire Somme (3 points)
Un *arbre binaire Somme* A est défini tel que pour tout nœud non feuille N de cet arbre la valeur de N est égale à la somme des valeurs de son fils droit et de son fils gauche. La valeur d'une feuille de A est fixée initialement.
Fournir l'algorithme récursif de la fonction **isBTSum** qui retourne vrai si l'arbre binaire spécifié est un *arbre binaire Somme*, faux sinon.

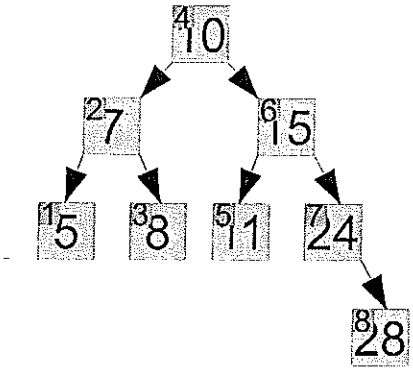
Exemple d'arbre binaire Somme :



Question 2 : Accès au k^{ième} élément d'un arbre binaire de recherche (4 points)
Soit un arbre binaire A contenant des valeurs entières strictement positives.
En supposant qu'on dispose d'une fonction **size(a : Arbre Binaire) : Entier** qui calcule le nombre de nœud de l'arbre A.
Fournir l'algorithme récursif de la fonction **findPos** permettant d'accéder au k^{ième} élément d'un ABR en suivant l'ordre croissant, si k≤0 ou k>size(a) alors findPos = -1

Exemple :
Soit a l'arbre ci-contre,

- si k=-6, findPos = -1
- si k=1, findPos = 5
- si k=4, findPos = 10
- si k=8, findPos = 28
- si k=15, findPos = -1



Exercice 3 : Automates (3 points)

Construire l'automate avec pour alphabet les chiffres de 0 à 9 (A=[0..9]) reconnaissant les nombres pairs.