Cours d'algorithmique et langage C

TDADS1150 – Les Arbres binaires et Arbres binaires de recherche ABR

On dispose des outils suivants :

Le type **Curseur**, qui permet de désigner un nœud de l'arbre. Un curseur peut être placé directement sur un nœud. Pour commencer, Il est généralement placé sur le nœud racine de l'arbre.

La fonction **obtenirUnCurseurSurLaRacine**(ArbreBinaire) qui renvoie un curseur sur la racine de l'arbre. Si la fonction **estEnFin**(curseur) renvoie vraie alors l'arbre est vide.

La fonction **descendreCurseur**(curseur, direction) permet déplacer le curseur dans la direction choisie, droite ou gauche, afin de se placer sur le fils droit ou le fils gauche.

La fonction **possèdeUnFils**(Curseur, direction) qui indique si le nœud désigné par le curseur possède un fils dans la direction choisie.

La fonction **insèreValeur**(Curseur, valeur) qui crée un nouveau nœud fils à l'endroit désigné par le curseur et y range la valeur (champ data).

La fonction **obtenirValeur**(Curseur) en Entier, qui renvoie la valeur contenue dans le nœud à l'endroit désigné par le curseur (champ data).

I. Ecrire le pseudo langage de la fonction **ajouterValeur**(arbre, valeur) qui recherche la position d'une nouvelle valeur à placer dans l'arbre ABR et insère cette valeur à l'aide de la fonction **insèreValeur**. La valeur à ajouter ne doit pas être déjà présente dans l'arbre.

Rappel : Dans un arbre ABR, on place la valeur sur le fils gauche si elle est inférieure ou sur le fils droit si elle est supérieure à la valeur du nœud visité. Si le nœud visité possède déjà un fils dans la direction choisie, on continue la recherche dans cette direction.

II. Les valeurs ont été ajoutées de manière organisée dans l'arbre ABR. On souhaite maintenant relire complètement l'arbre et **afficher les valeurs triées dans l'ordre croissant**. Pour cela il est possible de faire un parcours récursif de l'arbre en profondeur. En effet, on peut considérer que chaque nœud est la racine d'un sous-arbre. On peut trouver les valeurs inférieures dans le sous-arbre gauche, la valeur médiane dans la racine et les valeurs supérieures dans le sous-arbre droit. Donner le pseudo langage de cette fonction nommée **afficheSousArbreRécurs**(curseur).

nb : retrouver des explications sur le parcours des arbres en largeur ou en profondeur sur le serveur dev.isen.fr. Attention, les arbres présentés ne sont pas triés.

Le pseudo langage de ces deux fonctions tient sur une page. Ecrivez seul votre propre version dans un document Word et déposez le sur Campus en fin de séance.

Trouvez une aide à la rédaction de votre pseudo langage sur la page suivante.

Définition du type Arbre binaire :

```
typedef struct BinaryTree {
        BinaryTreeElement * anchor; // l'ancrage de l'arbre est réalisé à l'aide d'un noeud fictif dont on n'utilisera arbitrairement le lien vers fils gauche
        int nodeCount;                          // comme pointeur sur la racine de l'arbre. Si celui-ci est NULL alors l'arbre est vide.
} BinaryTree;                                   // nodeCount est une option qui permet de stocker le nombre de noeuds contenus dans cet arbre.
```

Ce qui donnerait en pseudo langage :

```
Déclaration
Type Arbre  en Enregistrement           // équivalent d'une structure en C
        Champ ancre             en Pointeur de Nœud
        Champ  nombreDeNoeud    en Entier
        fin d'Enregistrement
fin déclaration
```

Définition du type Noeud :

```
// élément d'un arbre binaire (noeud)
typedef struct BinaryTreeElement {
        int data;                       // le champ donnée d'un noeud, ici une valeur numérique entière
        int nodeID;                     // numéro d'identification d'un noeud. En option, pour des implémentations spécifiques
        struct BinaryTreeElement * leftChild; // pointeur sur le fils gauche
        struct BinaryTreeElement * rightChild; // pointeur sur le fils droit
} BinaryTreeElement;
```

Ce qui donnerait en pseudo langage :

```
Déclaration
Type Noeud  en Enregistrement           // équivalent d'une structure en C
        Champ data              en Entier
        Champ IDnoeud           en Entier
        Champ filsGauche        en Pointeur de Nœud
        Champ filsDroit         en Pointeur de Noeud
        fin d'Enregistrement
fin déclaration
```

Aide à la rédaction du pseudo langage : (programme complètement fictif)

```
Programme nom_programme

Fonction test1( a, b ) en Entier
Déclaration
        Constante Pi vaut 3.14
        Paramètre a, b          en Entier
        Variable tmp            en Entier
Début
        tmp <- (a +b) * Pi
        retourner(tmp)
fin fonction test1

Déclaration
        Variable nouveauNoeud           en Nœud
        Variable ptrNoeud               en pointeur de Nœud
        Variable nb                     en Entier
        Variable tab                    en tableau[10] d'Entier

Début
        ptrNoeud <- allouer(Nœud)    // ou <- nouveau Nœud
        Si (ptrNoeud <> AUCUNE_ADRESSE)  // ou NULL
        Alors
                (*ptrNoeud).filsDroit = AUCUNE_ADRESSE
                (*ptrNoeud).filsGauche = AUCUNE_ADRESSE
        FinSI
        Ecrire ("Entrez la valeur d'un nombre ")
        Lire(nb)
        Ecrire ("Le résultat est égal à ")
        tab[0]<- test1(nb, 2)
        Ecrire ( tab[0] )
        Si (nb > 0 )
        Alors
                Faire
                        Nb<-Nb - 1
                Tant Que ( NON ( nb = 0 ) )
        FinSi
Fin programme nom_programme.
```

ANNEXE : Aide à l'implémentation en langage C

```c
typedef enum Direction { Right, Left } Direction; // utilisé pour implémenter un curseur ou itérateur qui facilite la navigation dans l'arbre.


typedef struct Cursor{ // un curseur désigne le noeud enfant du côté 'side' du noeud parent référencé par le 'pointeur'
                                                    // par exemple, pour placer le curseur à la racine de l'arbre, il faut :
                                                    // initialiser le champ 'pointeur' avec la valeur de 'anchor' (le noeud d'ancrage)
                                                    // et initialiser 'side' avec la valeur 'Left'
                                                    // si l'arbre est vide alors la fonction estEnFin(curseur) renvoie 'true'.

        BinaryTreeElement* pointeur;
        Direction side;

} Cursor;



// renvoie un curseur sur le noeud à la racine de l'arbre
Cursor getCursorOnRoot(BinaryTree tree);

// renvoie un pointeur sur le noeud désigné par le curseur
BinaryTreeElement * getNode(Cursor curs);

// création d'un nouveau noeud d'arbre binaire
int newBinaryTreeElement(BinaryTreeElement ** node);

// création d'un nouvel arbre binaire
int newTree(BinaryTree ** tree);

// faire descendre un curseur dans l'arbre, à droite ou à gauche
void goDown(Cursor * curs, Direction dir);
```

```
// indique qu'il n'y a pas de fils dans la direction du curseur
// le curseur ne désigne pas un noeud
// on pourrait donc greffer (cf insertValueInAbrAt) un nouveau noeud
bool isAtTheEnd(Cursor curs);

// verifie que le noeud désigné par le cursor n'a pas de fils, et est donc une feuille
bool isALeaf(Cursor curs);

// vérifie que le noeud désigné par le cursor possède un successeur dans la direction indiquée
bool hasAChild(Cursor curs, Direction D);

// renvoie la valeur stockée dans le champ data d'un noeud
int getValue(Cursor curs);

// change la valeur contenue dans le champ data d'un noeud
void modifyValue(Cursor curs, int X);

// insere une nouvelle valeur dans une nouvelle feuille à la position indiquée par le curseur
// qui doit être en fin (cf estEnFin())
// utilisé par la fonction addValueToABR() pour ajouter une nouvelle valeur dans l'arbre
void insertValueInAbrAt(Cursor curs, int X);

// recherche l'extremité où on peut ajouter la valeur dans l'arbre ABR sauf si elle existe déjà
void addValueToABR(BinaryTree arbre, int x);
```