Structure de données TP6

Table des matières

[Exercice 1 2](#_Toc90403021)

[Algorithmes 2](#_Toc90403022)

[Code C 5](#_Toc90403023)

## Exercice 1

## Algorithmes

fonction insertTree(noeud\_t \*ptree, int val) : noeud\_t\*

début

    si ptree n'est pas null alors

        si ptree.val > result.val alors

            insertTree(ptree.fils\_gauche,val)

        sinon si ptree.val < result.val alors

            insertTree(ptree.fils\_droit,val)

            // sinon alors il y a égalité, donc pas besoin de nouvelle node

            fin\_si

        fin\_si

    sinon

        noeud\_t\* ptree = espace mémoire de taille noeud\_t

        ptree.val = val

    fin\_si

    retourner result

fin

fonction inorderTree(noeud\_t \*ptree) : void

début

    si ptree n'est pas null alors

        inorderTree(ptree.fils\_gauche)

        ecrire(ptree.val)

        inorderTree(ptree.fils\_droit)

    fin\_si

fin

fonction preorderTree(noeud\_t \*ptree) : void

début

    si ptree n'est pas null alors

        ecrire(ptree)

        preorderTree(ptree.fils\_gauche)

        preorderTree(ptree.fils\_droit)

    fin\_si

fin

fonction postorderTree(noeud\_t \*ptree) : void

début

    si ptree n'est pas null alors

        postorderTree(ptree.fils\_gauche)

        postorderTree(ptree.fils\_droit)

        ecrire(ptree)

    fin\_si

fin

fonction breadthTree(noeud\_t \*ptree) : void

début

    si ptree n'est pas null alors

        ecrire(ptree)

        si ptree.fils\_gauche != NULL alors ecrire(ptree.fils\_gauche)

        si ptree.fils\_droit != NULL alors ecrire(ptree.fils\_droit)

        reccu\_breadthTree(ptree.fils\_gauche)

        reccu\_breadthTree(ptree.fils\_droit)

    fin\_si

fin

fonction reccu\_breadthTree(noeud\_t \*ptree) : void

début // on n'écrit pas la node en question contrairement au début

      // pour le sommet

    si ptree n'est pas null alors

        si ptree.fils\_gauche != NULL alors ecrire(ptree.fils\_gauche)

        si ptree.fils\_droit != NULL alors ecrire(ptree.fils\_droit)

        reccu\_breadthTree(ptree.fils\_gauche)

        reccu\_breadthTree(ptree.fils\_droit)

    fin\_si

fin

fonction maxTree(noeud\_t \*ptree) : int

début

    noeud\_t\* p\_temp = ptree;

    si p\_temp n'est pas null alors

        tant que p\_temp.fils\_gauche n'est pas null faire

            p\_temp = p\_temp.fils\_gauche

        fin\_tant\_que

    sinon

        p\_temp.val = -1

    fin\_si

    retourner p\_temp.val

fin

fonction minTree(noeud\_t \*ptree) : int

début

    noeud\_t\* p\_temp = ptree;

    si p\_temp n'est pas null alors

        tant que p\_temp.fils\_droit n'est pas null faire

            p\_temp = p\_temp.fils\_droit

        fin\_tant\_que

    sinon

        p\_temp.val = -1

    fin\_si

    retourner p\_temp.val

fin

fonction heightTree(noeud\_t \*ptree) : int

début

    si ptree n'est pas null alors

        int val\_fils\_gauche = 1 + heightTree(ptree.fils\_gauche)

        int val\_fils\_droit = 1 + heightTree(ptree.fils\_droit)

        si val\_fils\_gauche >= val\_fils\_droit alors

            retourner val\_fils\_gauche

        sinon

            retourner val\_fils\_droit

        fin\_si

    sinon

        retourner 0

    fin\_si

fin

fonction nbNodesTree(noeud\_t \*ptree) : int

début

    si ptree n'est pas null alors

        int val\_fils\_gauche = 1 + nbNodesTree(ptree.fils\_gauche)

        int val\_fils\_droit = 1 + nbNodesTree(ptree.fils\_droit)

        retourner val\_fils\_droit + val\_fils\_gauche

    sinon

        retourner 0

    fin\_si

fin

fonction searchTree(noeud\_t \*ptree, int val) : noeud\_t\*

début

    noeud\_t\* res;

    si ptree n'est pas null alors

        si ptree.val == val alors

            res = ptree

        fin\_si

        res = searchTree(ptree.fils\_gauche,val)

        res = searchTree(ptree.fils\_droit,val)

    fin\_si

    retourner res

fin

fonction removeTree(noeud\_t \*ptree, int val) : noeud\_t\*

début

    noeud\_t p\_temp = searchTree(ptree,val)

    // impossible de retrouver le prédécesseur dans l'arbre

    // avec une méthode réccursive ici

    retourner p\_temp

fin

### Code C

#include "arbre.h"

void \*insertTree(noeud\_t \*ptree, int val) {

    if (ptree != NULL)

    {

        if (ptree->valeur > val)

        {

            insertTree(ptree->fils\_gauche,val);

        }

        else if (ptree->valeur < val)

        {

            insertTree(ptree->fils\_droit,val);

        }

    } else {

        ptree = malloc(sizeof(noeud\_t));

        ptree->valeur = val;

    }

}

void inorderTree(noeud\_t \*ptree) {

    if (ptree != NULL) {

        inorderTree(ptree->fils\_gauche);

        printf("%d /",ptree->valeur);

        inorderTree(ptree->fils\_gauche);

    }

}

void preorderTree(noeud\_t \*ptree) {

    if (ptree != NULL) {

        printf("%d /",ptree->valeur);

        preorderTree(ptree->fils\_gauche);

        preorderTree(ptree->fils\_gauche);

    }

}

void postorderTree(noeud\_t \*ptree) {

    if (ptree != NULL) {

        postorderTree(ptree->fils\_gauche);

        postorderTree(ptree->fils\_gauche);

        printf("%d /",ptree->valeur);

    }

}

void breadthTree(noeud\_t \*ptree) {

    if (ptree != NULL) {

        printf("%d /", ptree->valeur);

        if (ptree->fils\_gauche != NULL) {

            noeud\_t\* fg = ptree->fils\_gauche;

            printf("%d /", fg->valeur);

        }

        if (ptree->fils\_droit != NULL) {

            noeud\_t\* fd = ptree->fils\_droit;

            printf("%d /", fd->valeur);

        }

        reccu\_breadthTree(ptree->fils\_gauche);

        reccu\_breadthTree(ptree->fils\_droit);

    }

}

void reccu\_breadthTree(noeud\_t \*ptree) {

    if (ptree != NULL) {

        if (ptree->fils\_gauche != NULL) {

            noeud\_t\* fg = ptree->fils\_gauche;

            printf("%d /", fg->valeur);

        }

        if (ptree->fils\_droit != NULL) {

            noeud\_t\* fd = ptree->fils\_droit;

            printf("%d /", fd->valeur);

        }

        reccu\_breadthTree(ptree->fils\_gauche);

        reccu\_breadthTree(ptree->fils\_droit);

    }

}

int maxTree(noeud\_t \*ptree) {

    noeud\_t\* p\_temp = ptree;

    if (p\_temp != NULL) {

        while (p\_temp->fils\_gauche != NULL)

        {

            p\_temp = p\_temp->fils\_gauche;

        }

    } else {

        p\_temp->valeur = -1;

    }

    return p\_temp->valeur;

}

int minTree(noeud\_t \*ptree) {

    noeud\_t\* p\_temp = ptree;

    if (p\_temp != NULL) {

        while (p\_temp->fils\_droit != NULL)

        {

            p\_temp = p\_temp->fils\_droit;

        }

    } else {

        p\_temp->valeur = -1;

    }

    return p\_temp->valeur;

}

int heightTree(noeud\_t \*ptree) {

    if (ptree != NULL) {

        int val\_fils\_gauche = 1 + heightTree(ptree->fils\_gauche);

        int val\_fils\_droit = 1 + heightTree(ptree->fils\_droit);

        if (val\_fils\_gauche >= val\_fils\_droit) {

            return val\_fils\_gauche;

        } else {

            return val\_fils\_droit;

        }

    } else {

        return 0;

    }

}

int nbNodesTree(noeud\_t \*ptree) {

    if (ptree != NULL) {

        int val\_fils\_gauche = 1 + nbNodesTree(ptree->fils\_gauche);

        int val\_fils\_droit = 1 + nbNodesTree(ptree->fils\_droit);

        return val\_fils\_gauche + val\_fils\_droit;

    } else {

        return 0;

    }

}

noeud\_t \*searchTree(noeud\_t \*ptree, int val) {

    noeud\_t\* res;

    if (ptree != NULL) {

        if (ptree->valeur == val) {

            res = ptree;

        }

        res = searchTree(ptree->fils\_gauche,val);

        res = searchTree(ptree->fils\_droit,val);

    }

    return res;

}

noeud\_t \*removeTree(noeud\_t \*ptree, int val) ; // remove val from tree and return the new tree

int main() {

    noeud\_t \*myTree = NULL; // empty tree

    insertTree(myTree, 50);

    insertTree(myTree, 45);

    insertTree(myTree, 65);

    insertTree(myTree, 55);

    insertTree(myTree, 54);

    insertTree(myTree, 56);

    insertTree(myTree, 80);

    insertTree(myTree, 70);

    insertTree(myTree, 85);

    insertTree(myTree, 30);

    insertTree(myTree, 47);

    inorderTree(myTree, 0);

    preorderTree(myTree, 0);

    postorderTree(myTree, 0);

    printf("max = %d\n", maxTree(myTree));

    printf("min = %d\n", minTree(myTree));

    printf("nb of nodes = %d\n", nbNodesTree(myTree));

    printf("height tree = %d\n\n", heightTree(myTree));

    breadthTree(myTree);

    printf("search for 55 = %d\n", searchTree(myTree, 55)->valeur);

    noeud\_t \*pnd = searchTree(myTree, 77);

    printf("search for 77 = %p\n", pnd);

    myTree = removeTree(myTree, 65) ;

}