#define **\_CRT\_SECURE\_NO\_WARNINGS**

#include <stdio.h>

#include <stdlib.h>

#define **EXIT\_SUCCESS** 0

#define **MALLOC\_ERROR** -1

struct treeNode;

typedef struct treeNode\* TreePosition;

typedef struct treeNode {

    int data;

    TreePosition left;

    TreePosition right;

} TreeNode;

struct queueNode;

typedef struct queueNode\* QueuePosition;

typedef struct queueNode {

    TreePosition treeNode;

    QueuePosition next;

} QueueNode;

QueuePosition rear = NULL;

TreePosition **insert**(int x, TreePosition root);

TreePosition **search**(int x, TreePosition root);

int **printPreorder**(TreePosition root, int level);

int **printInorder**(TreePosition root, int level);

int **printPostorder**(TreePosition root, int level);

QueuePosition **enqueue**(TreePosition levelItem);

TreePosition **dequeue**();

int **printLevelorder**(TreePosition root);

TreePosition **findMin**(TreePosition root);

TreePosition **deleteElement**(TreePosition root, int x);

int **deleteTree**(TreePosition root);

int **main**()

{

    TreePosition root = NULL;

    root = **insert**(5, root);

    root = **insert**(3, root);

    root = **insert**(7, root);

    root = **insert**(2, root);

    root = **insert**(4, root);

    root = **insert**(1, root);

    root = **insert**(6, root);

    root = **insert**(8, root);

    root = **insert**(9, root);

    root = **insert**(10, root);

**search**(6, root);

**search**(25, root);

**printf**("Preorder ispis:\n");

**printPreorder**(root, 0);

**printf**("Inorder ispis:\n");

**printInorder**(root, 0);

**printf**("Postorder ispis:\n");

**printPostorder**(root, 0);

**printf**("Levelorder ispis:\n");

**printLevelorder**(root);

    root = **deleteElement**(root, 7);

    root = **deleteElement**(root, 6);

**printf**("Preorder ispis nakon brisanja elemenata 7 i 6:\n");

**printPreorder**(root, 0);

**printf**("Postorder nakon brisanja elemenata 7 i 6;\n");

**printPostorder**(root, 0);

**printf**("Inorder nakon brisanja elemenata 7 i 6;\n");

**printInorder**(root, 0);

**deleteTree**(root);

    return EXIT\_SUCCESS;

}

TreePosition **insert**(int x, TreePosition root)

{

    if (root == NULL)

    {

        root = (TreePosition)**malloc**(sizeof(TreeNode));

        if (root == NULL) {

**printf**("Malloc error.\n");

            return NULL;

        }

        root->data = x;

        root->left = NULL;

        root->right = NULL;

    }

    else if (x < root->data)

        root->left = **insert**(x, root->left);

    else if (x > root->data)

        root->right = **insert**(x, root->right);

    return root;

}

TreePosition **search**(int x, TreePosition root)

{

    if (root == NULL) {

**printf**("Nema *%d*.\n", x);

        return NULL;

    }

    if (x < root->data)

        return **search**(x, root->left);

    else if (x > root->data)

        return **search**(x, root->right);

**printf**("Postoji element *%d*.\n", x);

    return root;

}

int **printPreorder**(TreePosition root, int level)

{

    int i = 0;

    for (i; i < level; i++) **printf**("   ");

**printf**("*%d*\n", root->data);

    if (root->left != NULL) **printPreorder**(root->left, level + 1);

    if (root->right != NULL) **printPreorder**(root->right, level + 1);

    return 0;

}

int **printInorder**(TreePosition root, int level)

{

    int i = 0;

    if (root->left != NULL) **printInorder**(root->left, level + 1);

    for (i; i < level; i++) **printf**("   ");

**printf**("*%d*\n", root->data);

    if (root->right != NULL) **printInorder**(root->right, level + 1);

    return 0;

}

int **printPostorder**(TreePosition root, int level)

{

    int i = 0;

    if (root->left != NULL) **printPostorder**(root->left, level + 1);

    if (root->right != NULL) **printPostorder**(root->right, level + 1);

    for (i; i < level; i++) **printf**("   ");

**printf**("*%d*\n", root->data);

    return 0;

}

QueuePosition **enqueue**(TreePosition treeNode)

{

    QueuePosition queueNode;

    queueNode = (QueuePosition)**malloc**(sizeof(QueueNode));

    if (queueNode == NULL) {

**printf**("Malloc error.\n");

        return NULL;

    }

    queueNode->treeNode = treeNode;

    if (rear == NULL) {

        queueNode->next = queueNode;

        return queueNode;

    }

    queueNode->next = rear->next;

    rear->next = queueNode;

    rear = queueNode;

    return queueNode;

}

TreePosition **dequeue**()

{

    if (rear == NULL) {

        return NULL;

    }

    TreePosition levelItem;

    if (rear == rear->next) {

        levelItem = rear->treeNode;

**free**(rear);

        rear = NULL;

    }

    else {

        QueuePosition front = rear->next;

        levelItem = front->treeNode;

        rear->next = front->next;

**free**(front);

    }

    return levelItem;

}

int **printLevelorder**(TreePosition root)

{

    rear = **enqueue**(root);

    if (rear == NULL) return MALLOC\_ERROR;

    while (rear != NULL) {

        TreePosition treeNode = **dequeue**();

**printf**("*%d* ", treeNode->data);

        if (treeNode->left != NULL) rear = **enqueue**(treeNode->left);

        if (treeNode->right != NULL) rear = **enqueue**(treeNode->right);

    }

**printf**("\n");

    return 0;

}

TreePosition **findMin**(TreePosition root) {

    if (root->left == NULL) return root;

    return **findMin**(root->left);

}

TreePosition **deleteElement**(TreePosition root, int x)

{

    if (root == NULL) {

**printf**("Element not found.\n");

        return NULL;

    }

    if (x < root->data) root->left = **deleteElement**(root->left, x);

    else if (x > root->data) root->right = **deleteElement**(root->right, x);

    else if (root->left != NULL && root->right != NULL) {

        TreePosition minInRight = **findMin**(root->right);

        root->data = minInRight->data;

        root->right = **deleteElement**(root->right, root->data);

    }

    else {

        TreePosition temp = root;

        if (root->left == NULL) root = root->right;

        else root = root->left;

**free**(temp);

    }

    return root;

}

int **deleteTree**(TreePosition root) {

    if (root == NULL) return EXIT\_SUCCESS;

**deleteTree**(root->left);

**deleteTree**(root->right);

**free**(root);

    return EXIT\_SUCCESS;

}