**Tree – first page answers**

#include<stdio.h>

#include <malloc.h>

typedef struct BinNode {

int key;

BinNode\* left;

BinNode\* right;

}BinNode;

typedef struct listNode {

int key;

listNode\* children;

listNode\* next;

} listNode;

listNode\* newListNode(int key)

{

listNode\* node = (listNode\*)malloc(sizeof(listNode));

node->key = key; //same as above

node->children = NULL;

node->next = NULL;

return node;

}

BinNode\* newNode(int key) {

BinNode\* node = (BinNode\*)malloc(sizeof(BinNode));

node->key = key;

node->left = NULL;

node->right = NULL;

return node;

}

**1.**

int height(BinNode\* node) {

if (node == NULL)

return -1;

int r = height(node->right), l = height(node->left);

if (r > l)

return r + 1;

return l + 1;

}

**2.**

int sum(BinNode\* node) {

int sum1 = 0;

if (node == NULL)

return 0;

sum1 += node->key + sum(node->left) + sum(node->right);

return sum1;

}

**3.**

int biggest(BinNode\* node) {

int l = node->key;

int r = node->key;

if (node ->left != NULL)

l = biggest(node->left);

if (node->left != NULL)

r = biggest(node->right);

if (l > r)

return l;

if (r > l)

return r;

}

**4.**

int numShows(BinNode\* node, int x)

{

if (node == NULL)

return 0;

int sumShows = 0;

if (node->key == x)

sumShows++;

return sumShows += numShows(node->left, x) + numShows(node->right, x);

}

**5.**

**א.**

int isInTheTree(BinNode\* node, int x) {

if (node == NULL)

return 0;

if (node->key == x)

return 1;

int result = isInTheTree(node->left, x);

if (result)//אם היה כבר בחלק המשאלי ישר נחזיר 1 ולא נבדוק בכלל את החלק הימני

return 1;

result = isInTheTree(node->right, x);//זה יתבצע רק אם לא מצאנו בחלק השמאלי

return result;

}

**ב.**

int isInTheOtherTree(BinNode\* node1, BinNode\* node2) {

if (node2 == NULL)

return 1;

if (!isInTheTree(node1, node2->key))

return 0;

if (!isInTheOtherTree(node1, node2->left))

return 0;

if (!isInTheOtherTree(node1, node2->right))

return 0;

return 1;

}

**6.**

**א.**

int lessThenTree(BinNode\* node, int x) {

if (node == NULL)

return 1;

if (node->key <= x)

return 0;

if (!lessThenTree(node->left, x))

return 0;

if (!lessThenTree(node->right, x))

return 0;

return 1;

}

**ב.**

int treeLessThenTree(BinNode\* t1, BinNode\* t2) {// האם כל ערך בעץ 1 קטן מכל הערכים בעץ 2

if (t1 == NULL)

return 1;

if (!lessThenTree(t2, t1->key))

return 0;

if (!treeLessThenTree(t1->left, t2))

return 0;

if (!treeLessThenTree(t1->right, t2))

return 0;

return 1;

}

**7.**

int childsInRight(BinNode\* node) {

if (node == NULL)

return 0;

int sum = 0;

if (node->right != NULL)

sum++;

return sum += childsInRight(node->right) + childsInRight(node->left);

}

**8.**

int rankInListnode(listNode\* root) {

int rank = 0;

listNode\* node = root->children;

while (node != NULL)

{

rank++;

node = node->next;

}

int max = rank;

node = root->children;

while (node != NULL)

{

int rank2 = rankInListnode(node);

if (max < rank2)

max = rank2;

node = node->next;

}

return max;

}

**9.**

int heightOfUniversalTree(listNode\* node) {

if (node == NULL)

return -1;

int height = heightOfUniversalTree(node->children)+1;

int height2 = heightOfUniversalTree(node->next);

if (height > height2)

return height;

return height2;

}

**Main to check**

void main() {

//עץ בינארי

BinNode\* root1 = newNode(2);

root1->left = newNode(5);

root1->left->left = newNode(9);

root1->left->right = newNode(10);

root1->left->right->left = newNode(12);

root1->left->right->right = newNode(25);

root1->right = newNode(78);

root1->right->left = newNode(50);

root1->right->right = newNode(10);

root1->right->right->left = newNode(5);

root1->right->right->right = newNode(81);

BinNode\* root2 = newNode(2);

root2->left = newNode(5);

root2->left->left = newNode(9);

root2->left->right = newNode(10);

root2->left->right->left = newNode(12);

root2->left->right->right = newNode(25);

root2->right = newNode(78);

root2->right->left = newNode(50);

root2->right->right = newNode(10);

root2->right->right->left = newNode(5);

root2->right->right->right = newNode(81);

//עץ כללי

listNode\* root = newListNode(7);

root->children = newListNode(5);

root->children->next = newListNode(9);

root->children->next->next = newListNode(10);

root->children->children = newListNode(-4);

root->children->children->next = newListNode(-7);

root->children->children->next->next = newListNode(-5);

root->children->next->children = newListNode(3);

root->children->next->children->next = newListNode(7);

root->children->next->children->next->next = newListNode(2);

root->children->next->next->children = newListNode(19);

root->children->next->next->children->next = newListNode(14);

printf("%d\n", height(root1));

printf("%d\n", sum(root1));

printf("%d\n", biggest(root1));

printf("%d\n", numShows(root1,5));

printf("%d\n", isInTheTree(root1,12));

printf("%d\n", isInTheOtherTree(root1,root2));

printf("%d\n", lessThenTree(root1,79));

printf("%d\n", treeLessThenTree(root1,root2));

printf("%d\n", childsInRight(root1));

printf("%d\n", rankInListnode(root));

printf("%d\n", heightOfUniversalTree(root));

}