**מעבדה 9 – מת"מ -   
מגישים:  
אורי מלכא – 314862996  
אלן ציפין - 313206062  
  
שאלה 1 קוד  
פלט1:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#define true 1

#define false 0

#define Max(a,b) ((a) > (b)? (a): (b))

typedef struct

{

int key;

float val;

} Data;

typedef struct TreeNode

{

struct TreeNode\* left; //left child

struct TreeNode\* right; //right child

Data TreeNode\_info; //info data in node

} TreeNode;

typedef struct ListNode

{

struct ListNode\* next;

Data ListNode\_info; //info data in node

} ListNode;

// TREE Handling prototypes

int treeHight(TreeNode\*); // returns the hight of a given tree

float sum\_tree(TreeNode\*); //returns the sum of values on all tree nodes

void empty\_tree(TreeNode\*\*); // frees all the allocated memory and sets the root to NULL

TreeNode\* create\_tree\_node(Data);

void insert\_info\_by\_key(TreeNode\*\*, TreeNode\*\*, Data);

TreeNode\* create\_search\_tree();

// LIST Handling prototypes

void empty\_list(ListNode\*\*);// frees all the allocated memory and sets the head to NULL

ListNode\* create\_list\_node(Data);

void print\_list(ListNode\*); // Prints the elements of the list from head to tail

ListNode\* append\_lists(ListNode\*, ListNode\*);

// LIST <-> TREE conversions

ListNode\* tree\_to\_inorder\_list(TreeNode\*);

ListNode\* tree\_to\_preorder\_list(TreeNode\*);

int main()

{

TreeNode\* inputTree = NULL;

ListNode\* preList = NULL, \* inList = NULL;

printf("\n getting the input tree");

inputTree = create\_search\_tree(); // get the tree from user

printf("\nhight of tree is %d\n", treeHight(inputTree)); // print hight

printf("\nsum of values of tree is %f\n", sum\_tree(inputTree));

printf("\n printing input tree inorder");

inList = tree\_to\_inorder\_list(inputTree); // inorder traverse the tree

print\_list(inList);

printf("\n printing input tree preorder");

preList = tree\_to\_preorder\_list(inputTree); // preorder traverse the tree

print\_list(preList);

empty\_tree(&inputTree);

empty\_list(&preList);

empty\_list(&inList);

return 0;

}

int treeHight(TreeNode\* tree)// returns the hight of a given tree

{

if (tree == NULL) return -1;

return Max(treeHight(tree->left) + 1, treeHight(tree->right) + 1);

}

float sum\_tree(TreeNode\* tree) //returns the sum of values on all tree nodes

{

if (tree == NULL)

return 0;

return tree->TreeNode\_info.val + sum\_tree(tree->left) + sum\_tree(tree->right);

}

void empty\_tree(TreeNode\*\* root) // frees all the allocated memory and sets the root to NULL

{

if ((\*root) == NULL)

return;

empty\_tree(&((\*root)->left));

empty\_tree(&((\*root)->right));

free(\*root);

}

TreeNode\* create\_tree\_node(Data data)

{

TreeNode\* new\_tree = (TreeNode\*)malloc(sizeof(TreeNode));

if ((new\_tree) == NULL) {

printf("No memo");

return NULL; //Return null, insert\_info\_by\_key will free all the memory of the tree

}

new\_tree->TreeNode\_info.key = data.key;

new\_tree->TreeNode\_info.val = data.val;

new\_tree->left = NULL;

new\_tree->right = NULL;

}

void insert\_info\_by\_key(TreeNode\*\* root, TreeNode\*\* root\_original, Data data)

{

static int times;

if (!(\*root))

{

\*root = create\_tree\_node(data); //get new node for tree

if (times == 0)

\*root\_original = \*root;

if (\*root == NULL)

{

printf("Allocation tree node failed");

empty\_tree(root\_original); /\*deleting the whole tree,since root\_original is a pointer to the whole tree\*/

exit(1);

}

}

else

{

if (data.key < (\*root)->TreeNode\_info.key) //if the new node's key is smaller

insert\_info\_by\_key(&((\*root)->left), root\_original, data); //call insert\_node with the left child

else

insert\_info\_by\_key(&((\*root)->right), root\_original, data);

}

times++;

}

TreeNode\* create\_search\_tree()

{

Data infoNode;

char in\_char;

TreeNode\* tree = NULL;

do

{

printf("\n Do you want to enter another node? (y/n) :");

in\_char = \_getche();

if (in\_char == 'y')

{

printf("\nEnter key and value: ");

scanf("%d %f", &(infoNode.key), &(infoNode.val));

insert\_info\_by\_key(&tree, &tree, infoNode); /\*sending the same parameter twice for deleting a whole tree\*/

}

} while (in\_char == 'y');

return tree;

}

void empty\_list(ListNode\*\* head) // frees all the allocated memory and sets the head to NULL

{

if ((\*head) == NULL)

return;

empty\_list(&((\*head)->next));

free(\*head);

\*head = NULL;

}

ListNode\* create\_list\_node(Data data)

{

ListNode\* new\_node = (ListNode\*)malloc(sizeof(ListNode)); //allocate new node in heap

if (new\_node == NULL)

return NULL;

else

{

new\_node->ListNode\_info = data;

new\_node->next = NULL;

return new\_node; //returns intialized node

}

}

void print\_list(ListNode\* head)

{

if (head == NULL)

{

printf("\nThe list is EMPTY\n");

return;

}

printf("\nThe list is:\n");

while (head != NULL)

{

printf("\n (key=%d , val=%.01f)", head->ListNode\_info.key, head->ListNode\_info.val);

head = head->next;

}

printf("\n\n");

}

ListNode\* append\_lists(ListNode\* firstList, ListNode\* secondList)

{

ListNode\* res;

res = secondList; /\*if firstList is empty\*/

if (firstList)

{

res = firstList;

while (firstList->next)

firstList = firstList->next;

firstList->next = secondList;

}

return res;

}

ListNode\* tree\_to\_inorder\_list(TreeNode\* root)

{

ListNode\* curr = NULL, \* left = NULL, \* right = NULL;

if (root)

{

left = tree\_to\_inorder\_list(root->left);

right = tree\_to\_inorder\_list(root->right);

curr = create\_list\_node(root->TreeNode\_info);

curr->next = right;

left = append\_lists(left, curr);

}

return left;

}

ListNode\* tree\_to\_preorder\_list(TreeNode\* root)

{

ListNode\* curr = NULL, \* left = NULL, \* right = NULL;

if (root)

{

left = tree\_to\_inorder\_list(root->left);

right = tree\_to\_inorder\_list(root->right);

curr = create\_list\_node(root->TreeNode\_info);

curr->next = left;

left = append\_lists(curr, right);

}

return curr;

}

**פלט2:**

