**Data Structures**

**Spring 2020**

**Lab #3**

**Date: 2020. 04. 30**

**ID/Name: 201533661 이승수**

**[Lab #3] (60 points)**

**Generate a list of 198 random integers between 1 and 100.**

**Add ‘15’ and ’50’ to the list.**

**Create an AVL tree with the first (up to) 100 unique keys (including 15 and 50) using the AVL tree program provided.**

**Use inorder traversal for all of the following.**

**-Search the tree for keys 15, 20, 50,70, 90 (and print the result)**

**-Search and print the highest (max) key and print it.**

**-Search and print all keys < 15**

**-Search and print all keys > 50**

**-Search and print all keys between 15 and 50 (not including 15 or 50)**

**[Source Code]**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define RMAX 100

typedef struct node

{

int data;

struct node\* left, \* right;

int ht;

}node;

node\* insert(node\*, int);

node\* Delete(node\*, int);

void preorder(node\*);

void inorder(node\*);

int height(node\*);

node\* rotateright(node\*);

node\* rotateleft(node\*);

node\* RR(node\*);

node\* LL(node\*);

node\* LR(node\*);

node\* RL(node\*);

int BF(node\*);

void searchKey(node\* T,int key);

void searchHighestKey(node\* T);

node\* search(node\* T, int key);

void inorderLess(node\* T, int key);

void inorderMore(node\* T, int key);

void inorderBetween(node\* T, int key1, int key2);

void leftChild\_Traversal(node\* T);

void rightChild\_Traversal(node\* T);

int main()

{

node\* root = NULL;

int x, n, i, op;

/\*Specification 1/2\*/

/\*Generate 200 random numbers\*/

int randInt[200] = { 0, };

srand((unsigned)time(NULL)); //initialize

for (int t = 0; t < 198; t++) {

int random\_num = 1 + (int)rand() % RMAX;

randInt[t] = random\_num;

}

randInt[198] = 15;

randInt[199] = 50;

for (int t = 0; t < 200; t++) {

;// printf("%d ", randInt[t]);

}

/\*Insert into AVL Tree\*/

root = NULL;

for (int t = 0; t < 100; t++) {

root=insert(root,randInt[t]);

}

root = insert(root, 15);root = insert(root, 50);

inorder(root);

printf("\n");

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

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

/\*Search key: 15,20,50,70,90\*/

printf("[1. search key]\n");

searchKey(root, 15);

searchKey(root, 20);

searchKey(root, 50);

searchKey(root, 70);

searchKey(root, 90);

/\*Search highest key\*/

printf("[2. highest key]\n");

searchHighestKey(root);

/\*Search key<15\*/

printf("[3. key<15]\n");

inorderLess(root, 15);

/\*

leftChild\_Traversal(search(root, 15));

printf("\n");//

rightChild\_Traversal(search(root, 15));\*/

/\*Search key>50\*/

printf("\n[4. key>50]\n");

inorderMore(root, 50);

/\*Search 15<key<50\*/

printf("\n[5. 15<key<50]\n");

inorderBetween(root, 15,50);

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

do

{

printf("\n1)Create:");

printf("\n2)Insert:");

printf("\n3)Delete:");

printf("\n4)Print:");

printf("\n5)Quit:");

printf("\n\nEnter Your Choice:");

scanf\_s("%d", &op);

switch (op)

{

case 1: printf("\nEnter no. of elements:");

scanf\_s("%d", &n);

printf("\nEnter tree data:");

root = NULL;

for (i = 0; i < n; i++)

{

scanf\_s("%d", &x);

root = insert(root, x);

}

break;

case 2: printf("\nEnter a data:");

scanf\_s("%d", &x);

root = insert(root, x);

break;

case 3: printf("\nEnter a data:");

scanf\_s("%d", &x);

root = Delete(root, x);

break;

case 4: printf("\nPreorder sequence:\n");

preorder(root);

printf("\n\nInorder sequence:\n");

inorder(root);

printf("\n");

break;

}

} while (op != 5);

return 0;

}

/\*function: find key from avl Tree\*/

/\*input: T(root node address of avlTree), key(target data)\*/

/\*output: none(print key node's data,BF)\*/

void searchKey(node\* T, int key) {

if (T != NULL) {

searchKey(T->left, key);

if ((T->data) == key) {

printf("data(%d),BF(%d)\n", T->data, BF(T));

}

searchKey(T->right, key);

}

}

/\*function: search highest(most rightest) data at avlTree\*/

/\*input: root node address of avlTree\*/

/\*output: none(output from inorder())\*/

void searchHighestKey(node\* T) {

if (T != NULL) {

searchHighestKey(T->right);

if ((T->right) == NULL) {

printf("data(%d),BF(%d)\n", T->data, BF(T));

}

}

}

/\*function: search for node address with key data value\*/

/\*input: T(root node address), key(target data value)\*/

/\*output: node address which have target data value\*/

node\* search(node\* T, int key) {

node\* resultNode = NULL;

if (T != NULL) {

node\* result = NULL;

result = search(T->left, key);

if (result != NULL) {

resultNode = result;

}

if ((T->data) == key) {

//printf("data(%d),BF(%d)\n", T->data, BF(T));//print target key

resultNode = T;

//printf("result: %d\n",result->data);//ok

}

result = search(T->right, key);

if (result != NULL) {

resultNode = result;

}

return resultNode;

}

return resultNode;

}

/\*function: inorder traversal whose node's data value less than key\*/

/\*input: T(root node address), key(target data value)\*/

/\*output: none(output of inorder())\*/

void inorderLess(node\* T, int key) {

if (T != NULL)

{

inorderLess(T->left, key);

if ((T->data) < key) {

printf("%d(Bf=%d)", T->data, BF(T));

inorderLess(T->right, key);

}

}

}

/\*function: inorder traversal whose node's data value bigger than key\*/

/\*input: T(root node address), key(target data value)\*/

/\*output: none(output of inorder())\*/

void inorderMore(node\* T, int key) {

if (T != NULL)

{

inorderMore(T->left, key);

if ((T->data) > key) {

printf("%d(Bf=%d)", T->data, BF(T));

inorderMore(T->right, key);

}

}

}

/\*function: inorder traversal whose node's data value between than key1&key2\*/

/\*input: T(root node address), key1(smaller target data value), key2(bigger target data value)\*/

/\*output: none(output of inorder())\*/

void inorderBetween(node\* T, int key1, int key2) {

if (T != NULL)

{

inorderBetween(T->left, key1, key2);

if ((T->data) > key1 && (T->data) < key2) {

printf("%d(Bf=%d)", T->data, BF(T));

inorderBetween(T->right, key1, key2);

}

}

}

void leftChild\_Traversal(node\* T) {

inorder(T->left);

}

void rightChild\_Traversal(node\* T) {

inorder(T->right);

}

node\* insert(node\* T, int x)

{

if (T == NULL)

{

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

T->data = x;

T->left = NULL;

T->right = NULL;

}

else

if (x > T->data) // insert in right subtree

{

T->right = insert(T->right, x);

if (BF(T) == -2)

if (x > T->right->data)

T = RR(T);

else

T = RL(T);

}

else

if (x < T->data)

{

T->left = insert(T->left, x);

if (BF(T) == 2)

if (x < T->left->data)

T = LL(T);

else

T = LR(T);

}

T->ht = height(T);

return(T);

}

node\* Delete(node\* T, int x)

{

node\* p;

if (T == NULL)

{

return NULL;

}

else

if (x > T->data) // insert in right subtree

{

T->right = Delete(T->right, x);

if (BF(T) == 2)

if (BF(T->left) >= 0)

T = LL(T);

else

T = LR(T);

}

else

if (x < T->data)

{

T->left = Delete(T->left, x);

if (BF(T) == -2) //Rebalance during windup

if (BF(T->right) <= 0)

T = RR(T);

else

T = RL(T);

}

else

{

//data to be deleted is found

if (T->right != NULL)

{ //delete its inorder succesor

p = T->right;

while (p->left != NULL)

p = p->left;

T->data = p->data;

T->right = Delete(T->right, p->data);

if (BF(T) == 2)//Rebalance during windup

if (BF(T->left) >= 0)

T = LL(T);

else

T = LR(T); \

}

else

return(T->left);

}

T->ht = height(T);

return(T);

}

int height(node\* T)

{

int lh, rh;

if (T == NULL)

return(0);

if (T->left == NULL)

lh = 0;

else

lh = 1 + T->left->ht;

if (T->right == NULL)

rh = 0;

else

rh = 1 + T->right->ht;

if (lh > rh)

return(lh);

return(rh);

}

node\* rotateright(node\* x)

{

node\* y;

y = x->left;

x->left = y->right;

y->right = x;

x->ht = height(x);

y->ht = height(y);

return(y);

}

node\* rotateleft(node\* x)

{

node\* y;

y = x->right;

x->right = y->left;

y->left = x;

x->ht = height(x);

y->ht = height(y);

return(y);

}

node\* RR(node\* T)

{

T = rotateleft(T);

return(T);

}

node\* LL(node\* T)

{

T = rotateright(T);

return(T);

}

node\* LR(node\* T)

{

T->left = rotateleft(T->left);

T = rotateright(T);

return(T);

}

node\* RL(node\* T)

{

T->right = rotateright(T->right);

T = rotateleft(T);

return(T);

}

int BF(node\* T)

{

int lh, rh;

if (T == NULL)

return(0);

if (T->left == NULL)

lh = 0;

else

lh = 1 + T->left->ht;

if (T->right == NULL)

rh = 0;

else

rh = 1 + T->right->ht;

return(lh - rh);

}

void preorder(node\* T)

{

if (T != NULL)

{

printf("%d(Bf=%d)", T->data, BF(T));

preorder(T->left);

preorder(T->right);

}

}

void inorder(node\* T)

{

if (T != NULL)

{

inorder(T->left);

printf("%d(Bf=%d)", T->data, BF(T));

//printf("root(%d),left(%d),right(%d),ht(%d),BF(%d)\n", T->data, T->left, T->right, T->ht, BF(T));

inorder(T->right);

}

}

**[Result]**

**Trial 1:**



**Trial 2:**

