# Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

# Факультет інформатики та обчислювальної техніки Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 9 з дисципліни

«Основи програмування

»

Варіант 23

Виконав студент Панченко Сергій Віталійович

(шифр, прізвище, ім'я, по батькові)

Перевірив

( прізвище, ім'я, по батькові)

Київ 202

## Лабораторна робота 6

**Мета:** вивчити особливості організації обробки дерев

**Постановка задачі:**

**C:\Users\User\Desktop\ascreenshots\sdfhngtnrtntr.PNG**

C:

#include <iostream>

#include <list>

#include<vector>

using namespace std;

template<typename T1, typename T2>

class Tree {

private:

class TNode {

public:

TNode() {

TNode\* origin = nullptr;

TNode\* left = nullptr;

TNode\* right = nullptr;

}

T1 key;

T2 value;

TNode\* origin;

TNode\* left;

TNode\* right;

};

TNode\* root = nullptr;

TNode\* find\_node(T1 key) {

bool found = false;

TNode\* curr = this->root;

while (!found) {

if (key < curr->key) {

if (curr->left != nullptr) {

curr = curr->left;

}

//exceprion

}

else if (key > curr->key) {

if (curr->right != nullptr) {

curr = curr->right;

}

//exception

}

else if (key == curr->key) {

return curr;

}

}

}

list<TNode\*> roam(TNode\* start) {

list<TNode\*> nodes;

if (start->left != nullptr) {

nodes.push\_back(start->left);

list<TNode\*>res = this->roam(start->left);

for (auto i = res.begin(); i != res.end(); i++)

{

nodes.push\_back(\*i);

}

}

if (start->right != nullptr) {

nodes.push\_back(start->right);

list<TNode\*>res = this->roam(start->right);

for (auto i = res.begin(); i != res.end(); i++)

{

nodes.push\_back(\*i);

}

}

return nodes;

}

public:

Tree() {}

void insert(T1 key, T2 value) {

bool placed = false;

TNode\* curr = this->root;

//initialize root

if (this->root == nullptr) {

this->root = new TNode();

this->root->key = key;

this->root->value = value;

}

// if root initialized

else if (this->root != nullptr) {

while (!placed) {

if (key < curr->key) {

if (curr->left != nullptr) {

curr = curr->left;

}

else if (curr->left == nullptr) {

TNode\* newNode = new TNode();

curr->left = newNode;

newNode->origin = curr;

newNode->key = key;

newNode->value = value;

placed = true;

}

}

else if (key > curr->key) {

if (curr->right != nullptr) {

curr = curr->right;

}

else if (curr->right == nullptr) {

TNode\* newNode = new TNode();

curr->right = newNode;

newNode->origin = curr;

newNode->key = key;

newNode->value = value;

placed = true;

}

}

else if (key == curr->key) {

if (curr->left == nullptr) {

TNode\* newNode = new TNode();

curr->left = newNode;

newNode->origin = curr;

newNode->key = key;

newNode->value = value;

placed = true;

}

else if (curr->left != nullptr) {

curr = curr->left;

while (curr->right != nullptr) {

curr = curr->right;

}

TNode\* newNode = new TNode();

curr->right = newNode;

newNode->origin = curr;

newNode->key = key;

newNode->value = value;

placed = true;

}

}

}

}

}

void exclude(T1 key) {

TNode\* nodeToDel = this->find\_node(key);

if (nodeToDel->left != nullptr) {

TNode\* curr = nodeToDel->left;

while (curr->right != nullptr) {

curr = curr->right;

}

T1 temp = curr->key;

this->exclude(curr->key);

nodeToDel->key = temp;

}

else if (nodeToDel->left == nullptr) {

if (nodeToDel->right == nullptr){

if (nodeToDel->origin->right == nodeToDel)

nodeToDel->origin->right = nullptr;

else if (nodeToDel->origin->left == nodeToDel)

nodeToDel->origin->left = nullptr;

delete nodeToDel;

}

else if (nodeToDel->right != nullptr) {

TNode\* curr = nodeToDel->right;

while (curr->left != nullptr) {

curr = curr->left;

}

T1 temp = curr->key;

this->exclude(curr->key);

nodeToDel->key = temp;

}

}

}

list<pair<T1, T2>> to\_list() {

list<TNode\*> nodes = this->roam(this->root);

list<pair<T1, T2>> pairs;

pair<T1, T2> kv;

kv.first = this->root->key;

kv.second = this->root->value;

pairs.push\_back(kv);

for (auto i = nodes.begin() ; i != nodes.end() ; i++)

{

kv.first = (\*i)->key;

kv.second = (\*i)->value;

pairs.push\_back(kv);

}

return pairs;

}

static bool KeyUnique(pair<T1, T2> p1, pair<T1, T2> p2) {

return (p1.first == p2.first);

}

void merge(Tree t) {

list<pair<T1, T2>> l1 = this->to\_list();

list<pair<T1, T2>> l2 = t.to\_list();

list<TNode\*> nodes1 = this->roam(this->root);

list<TNode\*> nodes2 = t.roam(t.root);

//free the memory from old nodes

for (auto i = nodes1.begin(); i != nodes1.end(); i++)

{

delete (\*i);

}

this->root = nullptr;

//create list of unique keys

for (auto i = l2.begin(); i != l2.end(); i++)

{

bool present = false;

for (auto j = l1.begin(); j != l1.end(); j++)

{

if (i->first==j->first)

{

present = true;

}

}

if (!present)

{

l1.push\_back((\*i));

}

}

for (auto i = l1.begin(); i != l1.end(); i++)

this->insert(i->first, i->second);

}

};

void test1BuildTree() {

Tree<int, string> t1 = Tree<int, string>();

t1.insert(100, "q");

t1.insert(200, "w");

t1.insert(300, "e");

t1.insert(400, "r");

t1.insert(500, "t");

t1.insert(350, "y");

t1.insert(330, "u");

t1.insert(320, "i");

t1.insert(335, "o");

t1.insert(150, "p");

t1.insert(175, "a");

t1.insert(90, "s");

t1.insert(96, "d");

t1.insert(98, "f");

t1.insert(99, "g");

t1.insert(97, "h");

t1.insert(94, "j");

t1.insert(92, "k");

t1.insert(93, "l");

t1.insert(50, "z");

t1.insert(51, "x");

t1.insert(40, "c");

t1.insert(45, "v");

t1.insert(44, "b");

auto res = t1.to\_list();

for (auto i = res.begin(); i != res.end(); i++)

{

cout << i->first << "\t" << i->second << endl;

}

}

void test2Exclude() {

Tree<int, string> t1 = Tree<int, string>();

t1.insert(100, "q");

t1.insert(200, "w");

t1.insert(300, "e");

t1.insert(400, "r");

t1.insert(500, "t");

t1.insert(350, "y");

t1.insert(330, "u");

t1.insert(320, "i");

t1.insert(335, "o");

t1.insert(150, "p");

t1.insert(175, "a");

t1.insert(90, "s");

t1.insert(96, "d");

t1.insert(98, "f");

t1.insert(99, "g");

t1.insert(97, "h");

t1.insert(94, "j");

t1.insert(92, "k");

t1.insert(93, "l");

t1.insert(50, "z");

t1.insert(51, "x");

t1.insert(40, "c");

t1.insert(45, "v");

t1.insert(44, "b");

t1.exclude(96);

t1.exclude(100);

auto res = t1.to\_list();

for (auto i = res.begin(); i != res.end(); i++)

{

cout << i->first << "\t" << i->second << endl;

}

}

void test3UniteTrees(){

Tree<int, string>t1 = Tree<int, string>();

Tree<int, string>t2 = Tree<int, string>();

Tree<int, string>t3 = Tree<int, string>();

t1.insert(1, "a");

t1.insert(2, "b");

t1.insert(3, "c");

t1.insert(4, "d");

t1.insert(5, "e");

t1.insert(6, "f");

t1.insert(7, "g");

t1.insert(8, "h");

t2.insert(10, "z");

t2.insert(5, "x");

t2.insert(4, "c");

t2.insert(11, "v");

t2.insert(12, "b");

t2.insert(3, "n");

t2.insert(1, "m");

t2.insert(19, "k");

cout << "Tree 1" << endl;

auto res1 = t1.to\_list();

for (auto i = res1.begin(); i != res1.end(); i++)

{

cout << i->first << "\t" << i->second << endl;

}

cout << "Tree 2" << endl;

auto res2 = t2.to\_list();

for (auto i = res2.begin(); i != res2.end(); i++)

{

cout << i->first << "\t" << i->second << endl;

}

t1.merge(t2);

auto res3 = t1.to\_list();

cout << "Tree 3" << endl;

for (auto i = res3.begin(); i != res3.end(); i++)

{

cout << i->first << "\t" << i->second << endl;

}

}

int main()

{

cout << "Test1 Build tree" << endl;

test1BuildTree();

cout << "Test2 Exclude 100 nad 96" << endl;

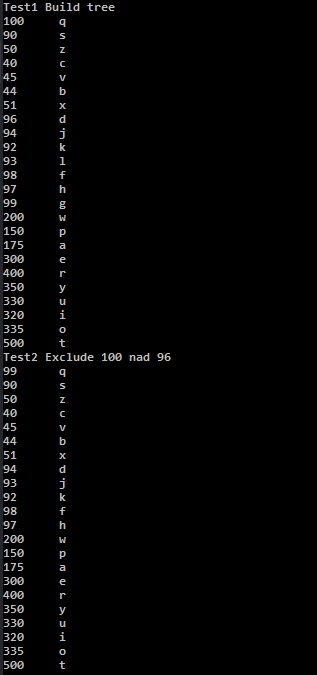
test2Exclude();

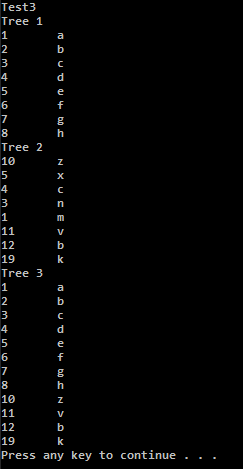
cout << "Test3" << endl;

test3UniteTrees();

}

**Результат:**

****

****

**Висновок:** під час лабораторної роботи навчилися організовувати роботу бінарних дерев. Оскільки програма створюює створює дерева, виконуючи умови поставленої до варіанту задачі, то лабораторна виконана правильно.