

#include <iostream>

#include <vector>

#include <Windows.h>

// Структура, якая прадстаўляе вузел дрэва

struct Node {

std::string str; // Дадзеныя ў вузле

int key; // Ключ для сартавання

Node\* left; // Паказальнік на левага нашчадка

Node\* right; // Паказальнік на правага нашчадка

// Канструктар для стварэння вузла з дадзенымі і ключом

Node(std::string s, int k) : str(s), left(nullptr), right(nullptr), key(k) {}

// Канструктар па змаўчанні

Node() : left(nullptr), right(nullptr) {}

};

// Функцыя дадання вузла ў дрэва

void addToTree(Node\*& root, const std::string& data, const int& key) {

if (root == nullptr) {

// Калі корань пусты, ствараем новы вузел і ён становіцца кораннем

root = new Node(data, key);

}

else if (key < root->key) {

// Калі ключ менш ключа кораня, рэкурсіўна дадаем у левае паддрэва

addToTree(root->left, data, key);

}

else {

// Інакш дадаем у правае паддрэва

addToTree(root->right, data, key);

}

}

// Функцыя абходу дрэва ў шырыню для атрымання дадзеных па ўзроўнях

void getLevelOrder(Node\* root, int level, std::vector<std::vector<std::string>>& res) {

if (root == nullptr) {

return;

}

if (level == res.size()) { // Ствараем і дадаем вектар для бягучага ўзроўню, калі мы апынуліся на гэтым узроўні ўпершыню(калі рэкурсіўная функцыя вернецца да выканання пераходу да правага паддрэва, level будзе менш памеру вектара .size() )

res.push\_back(std::vector<std::string>());

}

// Дадаем дадзеныя вузла ў бягучы ўзровень

res[level].push\_back(root->str);

// Рэкурсіўна абыходзім левае і правае паддрэвы

getLevelOrder(root->left, level + 1, res);

getLevelOrder(root->right, level + 1, res);

}

// Функцыя вываду вертыкальнага дрэва

void printTree(Node\* root) {

std::vector<std::vector<std::string>> res; // Вектар вектараў радкоў для элементаў дрэва па ўзроўнях

getLevelOrder(root, 0, res); // Запаўняем вектар

for (const std::vector<std::string>& level : res) { // Выкарыстоўваючы ітэратар праходзім па вектары, атрымліваючы вектары з радкамі, уяўлялыя ўзроўні дрэва

for (const std::string& node : level) { // Выводзім усе радкі кожнага ўзроўню

std::cout << node << " ";

}

std::cout << std::endl;

}

}

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

// Ствараем корань дрэва

Node\* root = nullptr;

// Даданне элементаў у дрэва

addToTree(root, "6", 6);

addToTree(root, "4", 4);

addToTree(root, "1", 1);

addToTree(root, "2", 2);

addToTree(root, "7", 7);

addToTree(root, "5", 5);

addToTree(root, "0", 0);

addToTree(root, "9", 9);

addToTree(root, "a", 10);

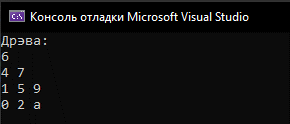
// Вывад вертыкальнага дрэва

std::cout << "Дрэва:" << std::endl;

printTree(root);

return 0;

}



Дап. Заданні



#include <iostream>

#include <vector>

#include <Windows.h>

// Структура, якая прадстаўляе вузел дрэва

struct Node {

std::string str; // Дадзеныя ў вузле

int key; // Ключ для сартавання

Node\* left; // Паказальнік на левага нашчадка

Node\* right; // Паказальнік на правага нашчадка

// Канструктар для стварэння вузла з дадзенымі і ключом

Node(std::string s, int k) : str(s), left(nullptr), right(nullptr), key(k) {}

// Канструктар па змаўчанні

Node() : left(nullptr), right(nullptr) {}

};

// Функцыя дадання вузла ў дрэва

void addToTree(Node\*& root, const std::string& data, const int& key) {

if (root == nullptr) {

// Калі корань пусты, ствараем новы вузел і ён становіцца кораннем

root = new Node(data, key);

}

else if (key < root->key) {

// Калі ключ менш ключа кораня, рэкурсіўна дадаем у левае паддрэва

addToTree(root->left, data, key);

}

else {

// Інакш дадаем у правае паддрэва

addToTree(root->right, data, key);

}

}

// Функцыя абходу дрэва ў шырыню для атрымання дадзеных па ўзроўнях

void getLevelOrder(Node\* root, int level, std::vector<std::vector<std::string>>& res) {

if (root == nullptr) {

return;

}

if (level == res.size()) { // Ствараем і дадаем вектар для бягучага ўзроўню, калі мы апынуліся на гэтым узроўні ўпершыню(калі рэкурсіўная функцыя вернецца да выканання пераходу да правага паддрэва, level будзе менш памеру вектара .size() )

res.push\_back(std::vector<std::string>());

}

// Дадаем дадзеныя вузла ў бягучы ўзровень

res[level].push\_back(root->str);

// Рэкурсіўна абыходзім левае і правае паддрэвы

getLevelOrder(root->left, level + 1, res);

getLevelOrder(root->right, level + 1, res);

}

// Функцыя вываду вертыкальнага дрэва

void printTree(Node\* root) {

std::vector<std::vector<std::string>> res; // Вектар вектараў радкоў для элементаў дрэва па ўзроўнях

getLevelOrder(root, 0, res); // Запаўняем вектар

for (const std::vector<std::string>& level : res) { // Выкарыстоўваючы ітэратар праходзім па вектары, атрымліваючы вектары з радкамі, уяўлялыя ўзроўні дрэва

for (const std::string& node : level) { // Выводзім усе радкі кожнага ўзроўню

std::cout << node << " ";

}

std::cout << std::endl;

}

}

// Функцыя для падліку і захавання галін з каранямі, ключы якіх больш або меньш за зададзенага ключа

void countRoots(Node\* root, int& count, const int& key, std::vector<std::vector<std::string>>& res) {

// Калі бягучы ўзровень не створаны, дадаем новы ўзровень у вынік

if (count == res.size()) {

res.push\_back(std::vector<std::string>());

}

// Калі корань пусты або яго ключ роўны зададзенаму ключу, вяртаемся

if (root == nullptr || root->key == key) {

return;

}

// Дадаем дадзеныя ў бягучы ўзровень

res[count++].push\_back(root->str);

// Калі ключ кораня больш зададзенага, рэкурсіўна абыходзім левае паддрэва

if (root->key > key) {

countRoots(root->left, count, key, res);

}

else { // Інакш правае паддрэва

countRoots(root->right, count, key, res);

}

}

// Функцыя для пошуку і вываду галін з каранямі, ключы якіх больш або меньш за зададзенага ключа

void find(Node\* root, int key) {

// Вектар для захоўвання вынікаў

std::vector<std::vector<std::string>> res;

// Лічыльнік для адсочвання бягучага ўзроўню, ўто адпавядае колькасці сустрэтых вузлоў

int count = 0;

// Выклікаем функцыю падліку галін

countRoots(root, count, key, res);

// Выводзім інфармацыю аб знойдзеных галінах

printf\_s("Галінак, ды вяршыні з ключом %d: %d\nГэтыя галіны:\n", key, count);

for (const auto& level : res) {

for (const auto& node : level) {

std::cout << node << " ";

}

std::cout << std::endl;

}

}

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

// Ствараем корань дрэва

Node\* root = nullptr;

// Даданне элементаў у дрэва

addToTree(root, "6", 6);

addToTree(root, "4", 4);

addToTree(root, "1", 1);

addToTree(root, "2", 2);

addToTree(root, "7", 7);

addToTree(root, "5", 5);

addToTree(root, "0", 0);

addToTree(root, "9", 9);

addToTree(root, "a", 10);

// Вывад вертыкальнага дрэва

std::cout << "Усё дрэва:" << std::endl;

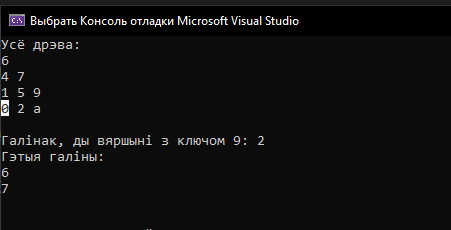
printTree(root);

std::cout << '\n';

find(root, 9);

return 0;

}





#include <iostream>

#include <vector>

#include <Windows.h>

// Структура, якая прадстаўляе вузел дрэва

struct Node {

std::string str; // Дадзеныя ў вузле

int key; // Ключ для сартавання

Node\* left; // Паказальнік на левага нашчадка

Node\* right; // Паказальнік на правага нашчадка

// Канструктар для стварэння вузла з дадзенымі і ключом

Node(std::string s, int k) : str(s), left(nullptr), right(nullptr), key(k) {}

// Канструктар па змаўчанні

Node() : left(nullptr), right(nullptr) {}

};

// Функцыя абходу дрэва ў шырыню для атрымання дадзеных па ўзроўнях

void getLevelOrder(Node\* root, int level, std::vector<std::vector<std::string>>& res) {

if (root == nullptr) {

return;

}

if (level == res.size()) { // Ствараем і дадаем вектар для бягучага ўзроўню, калі мы апынуліся на гэтым узроўні ўпершыню(калі рэкурсіўная функцыя вернецца да выканання пераходу да правага паддрэва, level будзе менш памеру вектара .size() )

res.push\_back(std::vector<std::string>());

}

// Дадаем дадзеныя вузла ў бягучы ўзровень

res[level].push\_back(root->str);

// Рэкурсіўна абыходзім левае і правае паддрэвы

getLevelOrder(root->left, level + 1, res);

getLevelOrder(root->right, level + 1, res);

}

// Функцыя вываду вертыкальнага дрэва

void printTree(Node\* root) {

std::vector<std::vector<std::string>> res; // Вектар вектараў радкоў для элементаў дрэва па ўзроўнях

getLevelOrder(root, 0, res); // Запаўняем вектар

for (const std::vector<std::string>& level : res) { // Выкарыстоўваючы ітэратар праходзім па вектары, атрымліваючы вектары з радкамі, уяўлялыя ўзроўні дрэва

for (const std::string& node : level) { // Выводзім усе радкі кожнага ўзроўню

std::cout << node << " ";

}

std::cout << std::endl;

}

}

void countLeaves(Node\* root, int& count) {

// Правяраем, ці не з'яўляецца бягучы вузел пустым

if (root == nullptr) {

return;

}

// Калі бягучы вузел з'яўляецца пялёсткам, павялічваем лічыльнік

if (root->left == nullptr && root->right == nullptr) {

count++;

return;

}

// Рэкурсіўна выклікаем функцыю для правага і левага паддрэваў

countLeaves(root->right, count);

countLeaves(root->left, count);

}

void countRightLeaves(Node\* root)

{

// калі дрэва няма, то і рабіць няма чаго

if (root == nullptr)

{

return;

}

int countRightLeaves = 0;

if (root->right == nullptr) // калі няма правага паддрэва

{

std::cout << "У дрэве няма правага паддрэва\n";

return;

}

countLeaves(root->right, countRightLeaves); // Інакш знаходзім палёсткі правага паддрэва, перадаючы яго

printf\_s("Колькасць палёсткаў у правым паддрэве: %d", countRightLeaves);

}

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

Node\* root = new Node("first", 1);

root->left = new Node("left 1", 2);

root->right = new Node("right 1", 4);

root->right->right = new Node("right 2", 3);

root->right->left = new Node("right 3", 8);

root->right->left->right = new Node("right 6", 8);

root->right->left->left = new Node("right 4", 8);

root->right->right->left = new Node("right 5", 8);

root->right->right->right = new Node("right 9", 8);

root->right->right->right->right = new Node("right 10", 8);

root->right->right->right->right->right = new Node("right 11", 8);

root->right->right->right->right->left = new Node("right 12", 8);

// Ствараем дрэва выгляду

//

// first(1)

// / \

// left 1(2) right 1(4)

// / \

// right 3(8) right 2(3)

// / \ / \

// right 4(8) right 6(8) right 5(8) right 9(8)

// \

// right 10(8)

// / \

// right 11(8) right 12(8)

std::cout << "Усё дрэва:" << std::endl;

printTree(root);

countRightLeaves(root);

return 0;

}



#include <iostream>

#include <vector>

#include <Windows.h>

// Структура, якая прадстаўляе вузел дрэва

struct Node {

std::string str; // Дадзеныя ў вузле

int key; // Ключ для сартавання

Node\* left; // Паказальнік на левага нашчадка

Node\* right; // Паказальнік на правага нашчадка

// Канструктар для стварэння вузла з дадзенымі і ключом

Node(std::string s, int k) : str(s), left(nullptr), right(nullptr), key(k) {}

// Канструктар па змаўчанні

Node() : left(nullptr), right(nullptr) {}

};

// Функцыя дадання вузла ў дрэва

void addToTree(Node\*& root, const std::string& data, const int& key) {

if (root == nullptr) {

// Калі корань пусты, ствараем новы вузел і ён становіцца кораннем

root = new Node(data, key);

}

else if (key < root->key) {

// Калі ключ менш ключа кораня, рэкурсіўна дадаем у левае паддрэва

addToTree(root->left, data, key);

}

else {

// Інакш дадаем у правае паддрэва

addToTree(root->right, data, key);

}

}

// Функцыя абходу дрэва ў шырыню для атрымання дадзеных па ўзроўнях

void getLevelOrder(Node\* root, int level, std::vector<std::vector<std::string>>& res) {

if (root == nullptr) {

return;

}

if (level == res.size()) { // Ствараем і дадаем вектар для бягучага ўзроўню, калі мы апынуліся на гэтым узроўні ўпершыню(калі рэкурсіўная функцыя вернецца да выканання пераходу да правага паддрэва, level будзе менш памеру вектара .size() )

res.push\_back(std::vector<std::string>());

}

// Дадаем дадзеныя вузла ў бягучы ўзровень

res[level].push\_back(root->str);

// Рэкурсіўна абыходзім левае і правае паддрэвы

getLevelOrder(root->left, level + 1, res);

getLevelOrder(root->right, level + 1, res);

}

// Функцыя вываду вертыкальнага дрэва

void printTree(Node\* root) {

std::vector<std::vector<std::string>> res; // Вектар вектараў радкоў для элементаў дрэва па ўзроўнях

getLevelOrder(root, 0, res); // Запаўняем вектар

for (const std::vector<std::string>& level : res) { // Выкарыстоўваючы ітэратар праходзім па вектары, атрымліваючы вектары з радкамі, уяўлялыя ўзроўні дрэва

for (const std::string& node : level) { // Выводзім усе радкі кожнага ўзроўню

std::cout << node << " ";

}

std::cout << std::endl;

}

}

void countNodes(Node\* root, int& count)

{

if (root == nullptr)

{

return;

}

// Рэкурсіўна абыходзім усе вяршыні і лічым іх

countNodes(root->left, count);

count++;

countNodes(root->right, count);

}

void countLeftNodes(Node\* root){

int c = 0;

countNodes(root->left, c);

std::cout << "Колькасць вяршыняў у левым поддереве: " << c << std::endl;

}

int main() {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

// Ствараем корань дрэва

Node\* root = nullptr;

// Даданне элементаў у дрэва

addToTree(root, "6", 6);

addToTree(root, "4", 4);

addToTree(root, "1", 1);

addToTree(root, "2", 2);

addToTree(root, "7", 7);

addToTree(root, "5", 5);

addToTree(root, "0", 0);

addToTree(root, "9", 9);

addToTree(root, "a", 10);

// Вывад вертыкальнага дрэва

std::cout << "Дрэва:" << std::endl;

printTree(root);

countLeftNodes(root);

return 0;

}

