24F-0040

Laiba

Lab 08

Problem 01:

#include<iostream>

#include<queue>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node() {}

Node(int a) {

data = a;

left = nullptr;

right = nullptr;

}

Node\* create(int a);

void invert(Node\* node);

void display(Node\* temp\_node);

};

Node\* root;

static Node\* create() {

int data;

cout << "Enter data to put in Tree(-1 for no node): ";

cin >> data;

if (data == -1) {

return nullptr;

}

Node\* new\_node = new Node(data);

cout << "\nEnter left child for: " << data;

new\_node->left = create();

cout << "\nEnter right child for: " << data;

new\_node->right = create();

return new\_node;

}

void Node::invert(Node\* root) {

if (root == nullptr) {

return;

}

Node\* temp = root->left;

root->left = root->right;

root->right = temp;

invert(root->left);

invert(root->right);

}

void Node::display(Node\* temp\_node) {

if (temp\_node == nullptr) {

return;

}

queue<Node\*> q;

q.push(temp\_node);

while (!q.empty()) {

Node\* crr = q.front();

q.pop();

cout << crr->data << " ";

if (crr->left != nullptr) {

q.push(crr->left);

}

if (crr->right != nullptr) {

q.push(crr->right);

}

}

}

int main() {

root = nullptr;

Node n;

root = create();

cout << "\nleft and right node swapping\n";

n.invert(root);

cout << "\n--------------------Display after swappiing--------------------\n";

n.display(root);

cout << endl;

system("Pause");

return 0;

}



Problem 02:

#include<iostream>

#include<queue>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node() {}

Node(int a) {

data = a;

left = nullptr;

right = nullptr;

}

Node\* create(int a);

void invert(Node\* node);

void display(Node\* temp\_node);

};

Node\* root;

static Node\* create() {

int data;

cout << "Enter data to put in Tree(-1 for no node): ";

cin >> data;

if (data == -1) {

return nullptr;

}

Node\* new\_node = new Node(data);

cout << "\nEnter left child for: " << data;

new\_node->left = create();

cout << "\nEnter right child for: " << data;

new\_node->right = create();

return new\_node;

}

void Node::invert(Node\* root) {

if (root == nullptr) {

return;

}

Node\* temp = root->left;

root->left = root->right;

root->right = temp;

invert(root->left);

invert(root->right);

}

void Node::display(Node\* temp\_node) {

if (temp\_node == nullptr) {

return;

}

queue<Node\*> q;

q.push(temp\_node);

while (!q.empty()) {

Node\* crr = q.front();

q.pop();

cout << crr->data << " ";

if (crr->left != nullptr) {

q.push(crr->left);

}

if (crr->right != nullptr) {

q.push(crr->right);

}

}

}

int main() {

root = nullptr;

Node n;

root = create();

cout << "\nleft and right node swapping\n";

n.invert(root);

cout << "\n--------------------Display after swappiing--------------------\n";

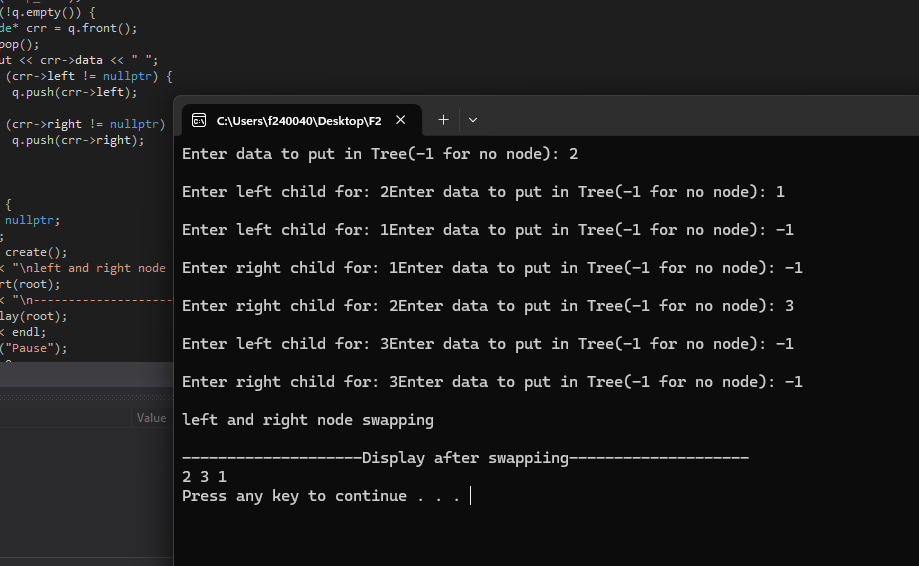
n.display(root);

cout << endl;

system("Pause");

return 0;

}



Problem 03:

#include<iostream>

#include<queue>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node() {}

Node(int a) {

data=a;

left = nullptr;

right = nullptr;

}

Node\* create();

void in\_order(Node\* root);

void pre\_order(Node\* root);

void output(Node\* root);

};

Node\* root;

static Node\* create() {

int data;

cout << "Enter data to add(-1 to add no data): ";

cin >> data;

if (data == -1) {

return nullptr;

}

Node\* new\_node = new Node(data);

cout << "Enter left child for: " << data << " ";

new\_node->left = create();

cout << "Enter right child for: " << data << " ";

new\_node->right = create();

return new\_node;

}

void Node::pre\_order(Node\* root) {

if (root == nullptr) {

return;

}

cout << root->data << " ";

pre\_order(root->left);

pre\_order(root->right);

}

void Node::in\_order(Node\* root) {

if (root == nullptr) {

return;

}

in\_order(root->left);

cout << root->data<<" ";

in\_order(root->right);

}

void Node::output(Node\* root) {

if (root == nullptr) {

return;

}

queue<Node\*>q;

q.push(root);

while (!q.empty()) {

Node\* crr = q.front();

q.pop();

if (crr == nullptr) {

cout << " NULL ";

continue;

}

cout << crr->data << " ";

q.push(crr->left);

q.push(crr->right);

}

}

int main() {

Node n;

root = nullptr;

root = create();

cout << "\nPre order Traversal: \n";

n.pre\_order(root);

cout << "\n In order Traversal: \n";

n.in\_order(root);

cout << "\n Output: \n";

n.output(root);

system("Pause");

return 0;

}

