**не предусмотрено удаление листа с заданным значением и есть лишнее**

#include <stdio.h>

#include <malloc.h>

#include <Windows.h>

#include <iostream>

using namespace std;

struct Grid

{

char\* Info;

int Key;

};

struct Tree

{

Grid field; // поле данных

Tree\* left; // левый потомок

Tree\* right; // правый потомок

};

void CheckInp(int\* number)

{

while (!scanf\_s("%d", number))

{

rewind(stdin);

cout <<"Некорректный формат ввода, повторите снова : "<<endl;

};

}

void CheckInpd(double\* number)

{

while (!scanf\_s("% lf", number))

{

rewind(stdin);

cout << "Некорректный формат ввода, повторите снова : " << endl;

};

}

char\* StringInp(char\* String)

{

rewind(stdin);

int i = 0, n = 1;

String = 0; -- серьезно??? Ничего не смущает???

do

{

String = (char\*)realloc(String, n++ \* sizeof(char));

String[i] = getchar();

i++;

} while (String[i - 1] != '\n');

String[i - 1] = 0;

return String;

}

void Sort(Grid\* elems, int size)

{

int counter;

Grid arrt;

for (int i = 1; i < size; i++)

{

arrt = elems[i];

counter = i - 1;

while (counter >= 0 && elems[counter].Key > arrt.Key)

{

elems[counter + 1] = elems[counter];

elems[counter] = arrt;

counter--;

}

}

}

Tree\* AddNode(Grid x, Tree\* root)

{

Tree\* tree = root;

if (tree == NULL)

{

tree = (Tree\*)calloc(1, sizeof(Tree));

tree -> field = x;

}

else if (x.Key < tree -> field.Key)

tree -> left = AddNode(x, tree -> left);

else

tree -> right = AddNode(x, tree -> right);

return tree;

}

void TreePrintPre(Tree\* tree)

{

if (tree != NULL)

{

cout << tree -> field.Info << endl;

cout << tree -> field.Key << endl;

TreePrintPre(tree -> left);

TreePrintPre(tree -> right);

}

}

void Find(Tree\* tree, int Key)

{

if (tree != NULL)

{

if (tree -> field.Key == Key)

{

cout << tree -> field.Info << endl;

cout << tree -> field.Key << endl;

}

Find(tree -> left, Key);

Find(tree -> right, Key);

}

}

int MaxValue(Tree\* node)

{

if (node == NULL)

return INT\_MIN;

int right = MaxValue(node -> right);

int left = MaxValue(node -> left);

if (node -> field.Key > left)

if (node -> field.Key > right)

return node -> field.Key;

else

return right;

else if (left < right)

return left;

else

return right;

}

int findMin(Tree\* root)

{

if (root == NULL)

return INT\_MAX;

int res = root -> field.Key;

int lres = findMin(root -> left);

int rres = findMin(root -> right);

if (lres < res)

res = lres;

if (rres < res)

res = rres;

return res;

}

void TreePrintIn(Tree\* tree)

{

if (tree != NULL)

{

TreePrintIn(tree -> left);

cout << tree -> field.Info << endl;

cout << tree -> field.Key << endl;

TreePrintIn(tree -> right);

}

}

void TreePrintPost(Tree\* tree)

{

if (tree != NULL)

{

TreePrintPost(tree -> left);

TreePrintPost(tree -> right);

cout << tree -> field.Info << endl;

cout << tree -> field.Key << endl;

}

}

void freemem(Tree\* tree) {

if (tree != NULL) {

freemem(tree -> left);

freemem(tree -> right);

delete tree;

}

}

Tree\* Delete(Tree\* root, int key)

{

Tree\* Del, \* Prev\_Del, \* R, \* Prev\_R;

// Del, Prev\_Del – удаляемый узел и его предыдущий (предок);

// R, Prev\_R – элемент, на который заменяется удаленный узел, и его предок;

Del = root;

Prev\_Del = NULL;

//-------- Поиск удаляемого элемента и его предка по ключу key ---------

while (Del != NULL && Del -> field.Key != key)

{

Prev\_Del = Del;

if (Del -> field.Key > key) Del = Del -> left;

else Del = Del -> right;

}

if (Del == NULL)

{ // Элемент не найден

cout << "NOT Key!\n " << endl;

return root;

}

//-------------------- Поиск элемента R для замены --------------------------------

if (Del -> right == NULL)

R = Del -> left;

else

if (Del -> left == NULL)

R = Del -> right;

else

{

//---------------- Ищем самый правый узел в левом поддереве -----------------

Prev\_R = Del;

R = Del -> left;

while (R -> right != NULL)

{

Prev\_R = R;

R = R -> right;

}

//----------- Нашли элемент для замены R и его предка Prev\_R -------------

if (Prev\_R == Del)

R -> right = Del -> right;

else

{

R -> right = Del -> right;

Prev\_R -> right = R -> left;

R -> left = Prev\_R;

}

}

if (Del == root)

root = R; // Удаляя корень, заменяем его на R

else

//------- Поддерево R присоединяем к предку удаляемого узла -----------

if (Del -> field.Key < Prev\_Del -> field.Key)

Prev\_Del -> left = R; // На левую ветвь

else

Prev\_Del -> right = R; // На правую ветвь

delete Del;

return root;

}

void Task(Tree\* root)

{

Tree\* MAXT, \* MINT;

MAXT = root;

MINT = root;

int min = findMin(root);

int max = MaxValue(root);

while (MAXT != NULL && MAXT -> field.Key != max)

{

if (MAXT -> field.Key > max) MAXT = MAXT -> left;

else MAXT = MAXT -> right;

}

while (MINT != NULL && MINT -> field.Key != min)

{

if (MINT -> field.Key > min) MINT = MINT -> left;

else MINT = MINT -> right;

}

cout << "Максимальное значение : "<< MAXT -> field.Info << " " << MAXT -> field.Key << endl;

cout << "Минимальное значение : " << MINT -> field.Info << " " << MINT -> field.Key << endl;

char\* stemp = nullptr;

int temp = 0, choice;

//-------------------- Поиск элемента R для замены --------------------------------

if (MAXT -> field.Key == max && MINT -> field.Key == min) {

stemp = MAXT -> field.Info;

temp = MAXT -> field.Key;

MAXT -> field.Info = MINT -> field.Info;

MAXT -> field.Key = MINT -> field.Key;

MINT -> field.Info = stemp;

MINT -> field.Key = temp;

}

cout << " 1.Префиксная\n2.Инфиксная\n3.Постфиксная\n " << endl;

CheckInp(&choice);

switch (choice)

{

case 1:

{

if (root)

TreePrintPre(root);

else

cout <<" Дерево пустое\n" << endl;

break;

}

case 2:

{

if (root)

TreePrintIn(root);

else

cout << " Дерево пустое\n" << endl; }

delete MAXT, MINT, stemp;

}

}

int main() {

Tree\* root = 0;

Grid\* StringGrid = 0;

int amount, choice, a = 1;

setlocale(LC\_ALL, "RUS");

while (a)

{

cout << " 1.Создание\n2.Просмотр\n3.Поиск поключу\n4.Персональное задание\n5.Удаление\n6.Выход\n" << endl;

cout << "Параметр: " << endl;

CheckInp(&a);

switch (a)

{

case 1:

{

if (!root)

{

cout << "Введите кол - во элементов : \n" << endl;

CheckInp(&amount);

StringGrid = (Grid\*)calloc(amount, sizeof(Grid));

for (int i = 0; i < amount; i++)

{

cout << " ФИО: " << endl;

StringGrid[i].Info = StringInp(StringGrid[i].Info);

cout << " Решенные примеры : " << endl;

CheckInp(&StringGrid[i].Key);

}

for (int i = 0; i < amount; i++)

root = AddNode(StringGrid[i], root);

}

else

cout << "Дерево уже создано\n" << endl;

break;

}

case 2:

{

cout << "1.Префиксная\n2.Инфиксная\n3.Постфиксная\n" << endl;

CheckInp(&choice);

switch (choice)

{

case 1:

{

if (root)

TreePrintPre(root);

else

cout << "Дерево пустое\n" << endl;

break;

}

case 2:

{

if (root)

TreePrintIn(root);

else

cout << "Дерево пустое\n" << endl;

break;

}

case 3:

{

if (root)

TreePrintPost(root);

else

cout << "Дерево пустое\n" << endl;

break;

}

default: break;

}

break;

}

case 3:

{

if (root)

{

int key;

cout << "Ключ: " << endl;

CheckInp(&key);

Find(root, key);

}

else

cout << " Дерево пустое\n" << endl;

break;

}

case 4:

{

Task(root);

break;

}

case 5:

{

int key;

if (root)

{

cout <<" Ключ: " << endl;

CheckInp(&key);

root = Delete(root, key);

amount--;

}

else

cout << "Дерево пустое\n" << endl;

break;

}

case 6:

{

freemem(root);

return 0;

}

}

}

}

if (d <= r->info)

{

// left

if (r->left != NULL)

return find(r->left, d); // рекурсивный поиск влево

else

{

return NULL; // не найден

}

}

else

{

//right

if (r->right)

return find(r->right, d);// рекурсивный поиск вправо

else

{

return NULL; // не найден

}

}

}

void treeprint(Tree\* tree)

{

if (tree != NULL) { //Пока не встретится пустой узел

treeprint(tree->left); //Рекурсивная функция вывода левого поддерева

cout << tree->info << endl; //Отображаем корень дерева

treeprint(tree->right); //Рекурсивная функция вывода правого поддерева

}

}

bool isLeaf(Tree\* node)

{ //Функция проверки, что узел дерева является листом

if (!node->left && !node->right)

{

return true;

}

return false;

}

int height(Tree\* node)

{ //Функция определения высоты дерева (число уровней)

if (node == NULL)

{

return 0;

}

else

{

int lheight = height(node->left);

int rheight = height(node->right);

if (lheight > rheight)

{

return (lheight + 1);

}

else

{

return (rheight + 1);

}

}

}

int printGivenLevel(Tree\* node, int level)

{ //Функция подсчета листьев на заданном уровне

if (node == NULL)

return 0;

static int k = 0;

if (level == 1)

{

if (isLeaf(node))

{

k++;

}

}

else if (level > 1)

{

printGivenLevel(node->left, level - 1);

printGivenLevel(node->right, level - 1);

}

return k;

}

void printLevelOrder(Tree\* root)

{

int h = height(root); //Количество уровней (высота дерева)

for (int i = 1; i <= h; i++)

{

cout << "Level #" << i << " => ";

int curr = printGivenLevel(root, i);

cout << printGivenLevel(root, i) - curr;

cout << "\n";

}

}

int main()

{

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

Tree\* root;

int tmp;

int\* ms = new int[10];

for (int i = 0; i < 10; i++)

{

ms[i] = unique\_el(ms, i);

}

for (int i = 0; i < 10; i++)

{

for (int i = 0; i < 9; i++)

{

if (ms[i] > ms[i + 1])

{

int tmp = ms[i];

ms[i] = ms[i + 1];

ms[i + 1] = tmp;

}

}

}

Make\_Blns(&root, 0, 10, ms);

cout << "Прямой обход сбалансированного дерева:\n";

Show(root);

cout << "Введите элемент, который вы хотите добавить: ";

int key;

cin >> key;

Add\_List(root, key);

cout << "Прямой обход сбалансированного дерева:\n";

Show(root);

cout << "Введите ключ, который вы хотите найти: ";

cin >> key;

if (find(root, key) == NULL) cout << "ключ не найден!";

else cout << "Найденный элемент: " << find(root, key)->info;

cout << "\nВведите ключ, который вы хотите удалить: ";

cin >> key;

Del\_Info(root, key);

cout << "Прямой обход сбалансированного дерева:\n";

Show(root);

cout << "Обратный обход сбалансированного дерева:\n";

Show\_rev(root);

cout << "Обход сбалансированного дерева по возрастанию:\n";

treeprint(root);

cout << "Кол-во листьев на каждом уровне:\n";

printLevelOrder(root); //Вывод числа листьев дерева по уровням

return 0;

}