#include <iostream>

#include <cstring>

#include <string>

#include <cstdint>

#include <cstdlib>

#include <Windows.h>

#pragma warning(disable:4996)

using namespace std;

struct Tree

{

int key = 0;

char str[1000] = { 0 };

Tree\* left = NULL, \* right = NULL;

};

Tree\* AddNode(Tree\*, int, char\*);

int DeleteNode(Tree\*\*, int);

void DeleteTree(Tree\*);

Tree\* FindNode(Tree\*, int key, Tree\*\*);

int TreeDepth(Tree\*);

void PrintTree(Tree const\*, int);

int FindLeaves(Tree\*);

int main()

{

SetConsoleOutputCP(1251); SetConsoleCP(1251);

Tree\* t = NULL;

Tree\* found = NULL;

int kod = 0, key = 0, kol = 0;

char str[1000]{};

while (true) {

cout << "\n"

"\tДобавить - 1.\n"

"\tПросмотр - 2.\n"

"\tЗадание - 3.\n"

"\tУдаление - 4.\n"

"\tВЫХОД - 0: ";

cin >> kod;

switch (kod) {

case 1:

cout << "\n\t\tКоличество узлов: ";

cin >> kol;

if (kol == 0) break;

while (kol < 0) {

cout << "\n\t\tОшибка: количество узлов не может быть меньше нуля\n\t\tВведите число : ";

cin >> kol;

}

for (int i = 0; i < kol; i++) {

cout << "\n\t\tКлюч узла: ";

cin >> key;

cout << "\t\tИнформация узла: ";

cin >> str;

t = AddNode(t, key, str);

}

if (t) {

if (kol == 1)

cout << "\n\t\tУзел успешно добавлен!\n";

if (kol > 0 && kol != 1)

cout << "\n\t\tУзлы успешно добавлены!\n";

}

break;

case 2:

if (t == NULL) {

cout << "\n\t\tПусто!\n";

break;

}

cout << "\n\t\tУзла - 1."

"\n\t\tДерева - 2: ";

cin >> kod;

switch (kod) {

case 1:

cout << "\n\t\t\tКлюч узла: ";

cin >> key;

found = FindNode(t, key, 0);

if (found == NULL)

cout << "\n\t\t\tУзел не найден\n";

else

{

cout << "\n\t\t\tКлюч: "

<< found->key

<< "\n\t\t\tИнформация: "

<< found->str << endl;

}

break;

case 2:

int depth = TreeDepth(t);

if (depth == 0) {

cout << "\n\t\t\tПусто!\n";

break;

}

cout << "\n\t\t\tГлубина дерева: " << depth << endl;

PrintTree(t, depth);

} break;

case 3: {

cout << "\n\t\tВведите ключ узла после которого считать листы: ";

cin >> key;

found = FindNode(t, key, 0);

if (found == NULL) {

cout << "\n\t\tУзел не найден\n";

break; }

int d = FindLeaves(found);

cout << "\n\t\tКоличество Листов: " << d << "\n";

} break;

case 4:

if (t == NULL) {

cout << "\n\t\tПусто!\n";

break;

}

cout << "\n\t\tУзла - 1."

"\n\t\tДерева - 2: ";

cin >> kod;

switch (kod) {

case 1:

cout << "\n\t\t\tКлюч: ";

cin >> key;

switch (DeleteNode(&t, key))

{

case -1:

cout << "\n\t\t\tУзел не найден\n";

break;

case 1:

cout << "\n\t\t\tПусто!\n";

break;

case 0:

cout << "\n\t\t\tУзел успешно удален!\n";

break;

default:

cout << "\n\t\t\t\t\t\tEgor, Why Two 'R'?\n"; //ERROR

return -1;

}

break;

case 2:

DeleteTree(t);

t = NULL;

cout << "\n\t\t\tПамять освобождена!\n";

} break;

case 0:

if (t) DeleteTree(t);

cout << "\n\t\t\t\t\t\tУСПЕХ!!!\n"; //Можно не писать, но успешны)

return 0;

}

}

}

Tree\* \_\_NewNode\_\_(int key, char\* str)

{

Tree\* ret = new Tree;

ret->key = key;

strcpy(ret->str, str);

return ret;

}

Tree\* AddNode(Tree\* t, int key, char\* str)

{

if (t == NULL)

{

t = \_\_NewNode\_\_(key, str);

return t;

}

Tree\* tmp = t;

while (1)

{

if (key < tmp->key)

{

if (tmp->left)

{

tmp = tmp->left;

continue;

}

tmp->left = \_\_NewNode\_\_(key, str);

return t;

}

if (tmp->right)

{

tmp = tmp->right;

continue;

}

tmp->right = \_\_NewNode\_\_(key, str);

return t;

}

return t;

}

int DeleteNode(Tree\*\* orig, int key)

{

if (orig == NULL)

return 1;

Tree\* p; // Родитель

Tree\* t = FindNode(\*orig, key, &p);

if (t == NULL) // Узел не найден

return -1;

if (t->left && t->right) // Два ребенка

{

char change\_orig = 0;

if (p == NULL)

change\_orig = 1;

Tree\* sp = t; // Поиск родителя

Tree\* s = t->left; // Поиск

while (s->right)

{

sp = s;

s = s->right;

}

if (change\_orig)

{

if (sp == \*orig)

{

s->right = (\*orig)->right;

delete(\*orig);

\*orig = s;

return 0;

}

sp->right = s->left;

s->right = (\*orig)->right;

s->left = (\*orig)->left;

delete(\*orig);

\*orig = s;

return 0;

}

if (sp == t)

{

if (p->left == t)

p->left = t->left;

else

p->right = t->left;

s->right = t->right;

delete(t);

return 0;

}

if (p->left == t)

p->left = s;

else

p->right = s;

sp->right = s->left;

s->right = t->right;

s->left = t->left;

delete(t);

return 0;

}

if (p) // Один ребенок

{

if (p->left == t)

p->left = (Tree\*)(((intptr\_t)t->left) + ((intptr\_t)t->right));

else

p->right = (Tree\*)(((intptr\_t)t->left) + ((intptr\_t)t->right));

delete(t);

}

else

{

Tree\* tmp = (Tree\*)(((intptr\_t)t->left) | ((intptr\_t)t->right));

delete(\*orig);

\*orig = tmp;

}

return 0;

}

void DeleteTree(Tree\* t)

{

if (t == NULL)

return;

DeleteTree(t->left);

DeleteTree(t->right);

delete t;

return;

}

Tree\* FindNode(Tree\* t, int key, Tree\*\* parent = NULL)

{

if (parent)

\*parent = NULL;

while (t)

{

if (t->key == key)

return t;

if (parent)

\*parent = t;

if (key < t->key)

{

t = t->left;

continue;

}

t = t->right;

}

return NULL;

}

int TreeDepth(Tree\* t) // Глубина дерева

{

if (t == NULL) {

return 0;

}

int l = TreeDepth(t->left);

int r = TreeDepth(t->right);

return 1 + (l > r ? l : r);

}

void PrintTree(Tree const\* t, int depth)

{

if (t == NULL)

return;

if (depth < 7)

depth = 7;

static char path[200]{};

static int len = 0;

printf

(

"\t\t\t%s%-\*c %-6d | %s\n",

len ? path : "Корень",

depth - (len ? len : 7), ':',

t->key, t->str

);

if (t->left)

{

path[len++] = 'Л';

PrintTree(t->left, depth);

path[len--] = 0;

}

if (t->right)

{

path[len++] = 'П';

PrintTree(t->right, depth);

path[len--] = 0;

}

return;

}

int FindLeaves(Tree\* t) // Функция задания

{

if (t == NULL)

return 0;

if (t->left == NULL && t->right == NULL)

return 1;

return FindLeaves(t->left) + FindLeaves(t->right);

}