Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа №11

По дисциплине «Основы алгоритмизации и программирования»

На тему «Бинарные деревья»

Выполнила:

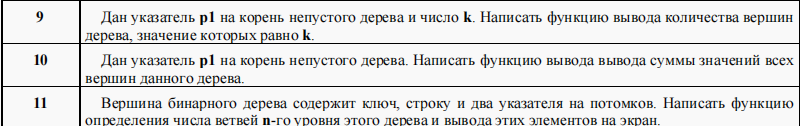
Студентка 1 курса 6 группы

Литвинчук Дарья Валерьевна

Преподаватель: асс. Андронова М.В.



Вариант 10



#include <iostream>

#include <windows.h>

using namespace std;

struct Tree //дерево

{

int key; //ключ

Tree\* Left, \* Right;

};

Tree\* makeTree(Tree\* Root); //Создание дерева

Tree\* list(int i); //Создание нового элемента

Tree\* insertElem(Tree\* Root, int key); //Добавление нового элемента

Tree\* search(Tree\* n, int key); //Поиск элемента по ключу

Tree\* delet(Tree\* Root, int key); //Удаление элемента по ключу

int view(Tree\* t, int level); //Вывод дерева

bool delAll(Tree\* t); //Очистка дерева

int getRootSum(Tree\* t);

int sum = 0;

Tree\* Root = NULL; //указатель на корень

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int key, choice;

Tree\* rc;

for (;;)

{

cout << "1 - создание дерева\n";

cout << "2 - добавление элемента\n";

cout << "3 - поиск по ключу\n";

cout << "4 - удаление элемента\n";

cout << "5 - вывод дерева\n";

cout << "6 - очистка дерева\n";

cout << "7 - вывести сумму всех вершин дерева\n";

cout << "0 - выход\n";

cout << "ваш выбор?\n";

cin >> choice;

cout << "\n";

switch (choice) {

case 1: {

Root = makeTree(Root);

break;

}

case 2: {

cout << "\nВведите ключ: ";

cin >> key;

insertElem(Root, key);

break;

}

case 3: {

cout << "\nВведите ключ: ";

cin >> key;

rc = search(Root, key);

break;

}

case 4: {

cout << "\nВведите удаляемый ключ: ";

cin >> key;

Root = delet(Root, key);

break;

}

case 5: {

if (Root->key >= 0) {

cout << "Дерево повернуто на 90 град. влево" << endl;

view(Root, 0);

}

else {

cout << "Дерево пустое\n";

break;

}

}

case 6: {

delAll(Root);

break;

}

case 7: {

cout << "Сумма вершин в дереве: " << getRootSum(Root) << endl;

break;

}

case 0: {

exit(0);

}

}

}

return 0;

}

Tree\* makeTree(Tree\* Root) { //Создание дерева

int key;

cout << "Конец ввода - отрицательное число\n\n";

if (Root == NULL) { // если дерево не создано

cout << "Введите ключ корня: ";

cin >> key;

Root = list(key); // установка указателя на корень

}

while (1) { //добавление элементов

cout << "\nВведите ключ: ";

cin >> key;

if (key < 0) {

break; //признак выхода (ключ < 0)

}

insertElem(Root, key);

}

return Root;

}

Tree\* list(int i) //Создание нового элемента

{

Tree\* t = new Tree[sizeof(Tree)];

t->key = i;

t->Left = t->Right = NULL;

return t;

}

Tree\* insertElem(Tree\* t, int key) //Добавление нового элемента

{

Tree\* Prev; // Prev - элемент перед текущим

Prev = t;

int find = 0; // признак поиска

while (t && !find) {

if (key == t->key) {

find = 1; //ключи должны быть уникальны

}

else {

if (key < t->key) {

t = t->Left;

}

else {

t = t->Right;

}

}

}

if (!find) { //найдено место с адресом Prev

t = list(key); //создается новый узел

if (key < Prev->key) { // и присоединяется либо

Prev->Left = t; //переход на левую ветвь,

}

else {

Prev->Right = t; // либо на правую

}

}

return t;

}

Tree\* delet(Tree\* Root, int key) //Удаление элемента по ключу

{ // Del, Prev\_Del - удаляемый элемент и его предыдущий;

// R, Prev\_R - элемент, на который заменяется удаленный, и его родитель;

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

Del = Root;

Prev\_Del = NULL;

while (Del != NULL && Del->key != key)//поиск элемента и его родителя

{

Prev\_Del = Del;

if (Del->key > key)

Del = Del->Left;

else

Del = Del->Right;

}

if (Del == NULL) // элемент не найден

{

puts("\nНет такого ключа");

return Root;

}

if (Del->Right == NULL) // поиск элемента R для замены

R = Del->Left;

else

if (Del->Left == NULL)

R = Del->Right;

else

{

Prev\_R = Del; //поиск самого правого элемента в левом поддереве

R = Del->Left;

while (R->Right != NULL)

{

Prev\_R = R;

R = R->Right;

}

if (Prev\_R == Del) // найден элемент для замены R и его родителя Prev\_R

R->Right = Del->Right;

else

{

R->Right = Del->Right;

Prev\_R->Right = R->Left;

R->Left = Prev\_R;

}

}

if (Del == Root) Root = R; //удаление корня и замена его на R

else

// поддерево R присоединяется к родителю удаляемого узла

if (Del->key < Prev\_Del->key)

Prev\_Del->Left = R; //на левую ветвь

else

Prev\_Del->Right = R; //на правую ветвь

int tmp = Del->key;

cout << "\nУдален элемент с ключом " << tmp << endl;

delete Del;

return Root;

}

Tree\* search(Tree\* n, int key) { //Поиск элемента по ключу

Tree\* rc = n;

if (rc != NULL) {

if (key < (key, n->key)) {

rc = search(n->Left, key);

}

else

if (key > (key, n->key)) {

rc = search(n->Right, key);

}

}

else

cout << "Нет такого элемента\n";

return rc;

}

int view(Tree\* t, int level) //Вывод дерева

{

if (t) {

view(t->Right, level + 1); //вывод правого поддерева

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

cout << " ";

}

int tm = t->key;

cout << tm << ' ';

view(t->Left, level + 1); //вывод левого поддерева

return 0;

}

return 1;

}

bool delAll(Tree\* t) { //Очистка дерева

if (t != NULL) {

delAll(t->Left);

delAll(t->Right);

delete t;

return true;

}

return false;

}

int getRootSum(Tree\* t) {

if (t == nullptr) { //Если дерево пустое

return 0;

}

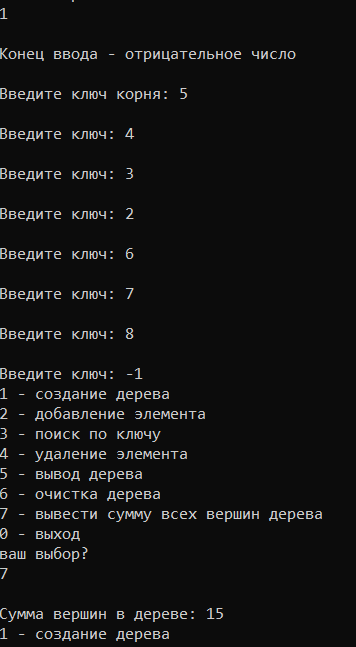
sum += t->key; //Находим сумму всех вершин

getRootSum(t->Left); //Просматриваем левую ветвь

getRootSum(t->Right); //Затем правую

return sum;

}



Доп. задания

Вариант 7



#include <iostream>

#include <windows.h>

using namespace std;

struct Tree //дерево

{

int key; //ключ

Tree\* Left, \* Right;

};

Tree\* makeTree(Tree\* Root); //Создание дерева

Tree\* list(int i); //Создание нового элемента

Tree\* insertElem(Tree\* Root, int key); //Добавление нового элемента

Tree\* search(Tree\* n, int key); //Поиск элемента по ключу

Tree\* delet(Tree\* Root, int key); //Удаление элемента по ключу

int view(Tree\* t, int level); //Вывод дерева

bool delAll(Tree\* t); //Очистка дерева

int nLeft = 0, nRight = 0; //Подсчет количество листьев слева и справа

int getLeavesAmount(Tree\* t);

Tree\* Root = NULL; //указатель на корень

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int key, choice;

Tree\* rc;

for (;;)

{

cout << "1 - создание дерева\n";

cout << "2 - добавление элемента\n";

cout << "3 - поиск по ключу\n";

cout << "4 - удаление элемента\n";

cout << "5 - вывод дерева\n";

cout << "6 - очистка дерева\n";

cout << "7 - вывести количество листьев дерева\n";

cout << "0 - выход\n";

cout << "ваш выбор?\n";

cin >> choice;

cout << "\n";

switch (choice) {

case 1: {

Root = makeTree(Root);

break;

}

case 2: {

cout << "\nВведите ключ: ";

cin >> key;

insertElem(Root, key);

break;

}

case 3: {

cout << "\nВведите ключ: ";

cin >> key;

rc = search(Root, key);

break;

}

case 4: {

cout << "\nВведите удаляемый ключ: ";

cin >> key;

Root = delet(Root, key);

break;

}

case 5: {

if (Root->key >= 0) {

cout << "Дерево повернуто на 90 град. влево" << endl;

view(Root, 0);

}

else {

cout << "Дерево пустое\n";

break;

}

}

case 6: {

delAll(Root);

break;

}

case 7: {

cout << "Количество листьев в дереве: " << getLeavesAmount(Root) << endl;

break;

}

case 0: {

exit(0);

}

}

}

return 0;

}

Tree\* makeTree(Tree\* Root) { //Создание дерева

int key;

cout << "Конец ввода - отрицательное число\n\n";

if (Root == NULL) { // если дерево не создано

cout << "Введите ключ корня: ";

cin >> key;

Root = list(key); // установка указателя на корень

}

while (1) { //добавление элементов

cout << "\nВведите ключ: ";

cin >> key;

if (key < 0) {

break; //признак выхода (ключ < 0)

}

insertElem(Root, key);

}

return Root;

}

Tree\* list(int i) //Создание нового элемента

{

Tree\* t = new Tree[sizeof(Tree)];

t->key = i;

t->Left = t->Right = NULL;

return t;

}

Tree\* insertElem(Tree\* t, int key) //Добавление нового элемента

{

Tree\* Prev; // Prev - элемент перед текущим

Prev = t;

int find = 0; // признак поиска

while (t && !find) {

if (key == t->key) {

find = 1; //ключи должны быть уникальны

}

else {

if (key < t->key) {

t = t->Left;

}

else {

t = t->Right;

}

}

}

if (!find) { //найдено место с адресом Prev

t = list(key); //создается новый узел

if (key < Prev->key) { // и присоединяется либо

Prev->Left = t; //переход на левую ветвь,

}

else {

Prev->Right = t; // либо на правую

}

}

return t;

}

Tree\* delet(Tree\* Root, int key) //Удаление элемента по ключу

{ // Del, Prev\_Del - удаляемый элемент и его предыдущий;

// R, Prev\_R - элемент, на который заменяется удаленный, и его родитель;

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

Del = Root;

Prev\_Del = NULL;

while (Del != NULL && Del->key != key)//поиск элемента и его родителя

{

Prev\_Del = Del;

if (Del->key > key)

Del = Del->Left;

else

Del = Del->Right;

}

if (Del == NULL) // элемент не найден

{

puts("\nНет такого ключа");

return Root;

}

if (Del->Right == NULL) // поиск элемента R для замены

R = Del->Left;

else

if (Del->Left == NULL)

R = Del->Right;

else

{

Prev\_R = Del; //поиск самого правого элемента в левом поддереве

R = Del->Left;

while (R->Right != NULL)

{

Prev\_R = R;

R = R->Right;

}

if (Prev\_R == Del) // найден элемент для замены R и его родителя Prev\_R

R->Right = Del->Right;

else

{

R->Right = Del->Right;

Prev\_R->Right = R->Left;

R->Left = Prev\_R;

}

}

if (Del == Root) Root = R; //удаление корня и замена его на R

else

// поддерево R присоединяется к родителю удаляемого узла

if (Del->key < Prev\_Del->key)

Prev\_Del->Left = R; //на левую ветвь

else

Prev\_Del->Right = R; //на правую ветвь

int tmp = Del->key;

cout << "\nУдален элемент с ключом " << tmp << endl;

delete Del;

return Root;

}

Tree\* search(Tree\* n, int key) { //Поиск элемента по ключу

Tree\* rc = n;

if (rc != NULL) {

if (key < (key, n->key)) {

rc = search(n->Left, key);

}

else

if (key > (key, n->key)) {

rc = search(n->Right, key);

}

}

else

cout << "Нет такого элемента\n";

return rc;

}

int view(Tree\* t, int level) //Вывод дерева

{

if (t) {

view(t->Right, level + 1); //вывод правого поддерева

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

cout << " ";

}

int tm = t->key;

cout << tm << ' ';

view(t->Left, level + 1); //вывод левого поддерева

return 0;

}

return 1;

}

bool delAll(Tree\* t) { //Очистка дерева

if (t != NULL) {

delAll(t->Left);

delAll(t->Right);

delete t;

return true;

}

return false;

}

int getLeavesAmount(Tree\* t) {

if (t == nullptr) { //Если дерево пустое

return 0;

}

if (t->Left == NULL && t->Right == NULL) {

return 1;

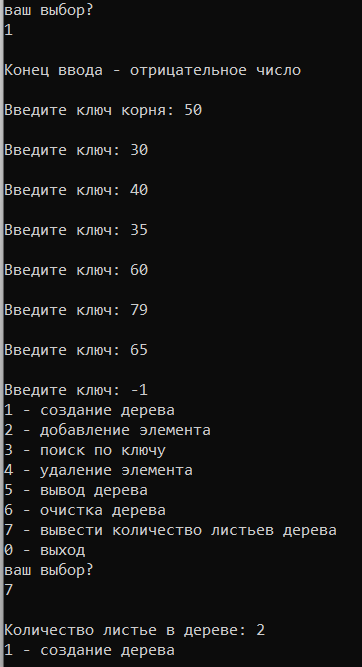
}

nLeft += getLeavesAmount(t->Left); //Просматриваем левую ветвь

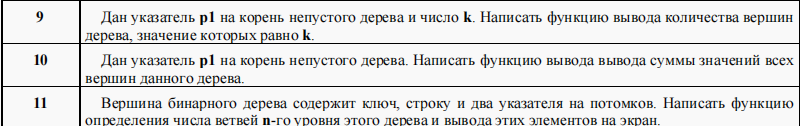
nRight += getLeavesAmount(t->Right); //Затем правую

return nLeft + nRight;

}



Вариант 9



#include <iostream>

#include <windows.h>

using namespace std;

struct Tree //дерево

{

int key; //ключ

Tree\* Left, \* Right;

};

Tree\* makeTree(Tree\* Root); //Создание дерева

Tree\* list(int i); //Создание нового элемента

Tree\* insertElem(Tree\* Root, int key); //Добавление нового элемента

Tree\* search(Tree\* n, int key); //Поиск элемента по ключу

Tree\* delet(Tree\* Root, int key); //Удаление элемента по ключу

int view(Tree\* t, int level); //Вывод дерева

bool delAll(Tree\* t); //Очистка дерева

int getCoincidences(Tree\* t, int k, int amount);

int k, amount = 0;

Tree\* Root = NULL; //указатель на корень

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int key, choice;

Tree\* rc;

for (;;)

{

cout << "1 - создание дерева\n";

cout << "2 - добавление элемента\n";

cout << "3 - поиск по ключу\n";

cout << "4 - удаление элемента\n";

cout << "5 - вывод дерева\n";

cout << "6 - очистка дерева\n";

cout << "7 - найти количество вершин, значение которых равно k\n";

cout << "0 - выход\n";

cout << "ваш выбор?\n";

cin >> choice;

cout << "\n";

switch (choice) {

case 1: {

Root = makeTree(Root);

break;

}

case 2: {

cout << "\nВведите ключ: ";

cin >> key;

insertElem(Root, key);

break;

}

case 3: {

cout << "\nВведите ключ: ";

cin >> key;

rc = search(Root, key);

break;

}

case 4: {

cout << "\nВведите удаляемый ключ: ";

cin >> key;

Root = delet(Root, key);

break;

}

case 5: {

if (Root->key >= 0) {

cout << "Дерево повернуто на 90 град. влево" << endl;

view(Root, 0);

}

else {

cout << "Дерево пустое\n";

break;

}

}

case 6: {

delAll(Root);

break;

}

case 7: {

cout << "Введите число k: ";

cin >> k;

cout << "количество вершин, значение которых равно k: " << getCoincidences(Root, k, amount) << endl;

break;

}

case 0: {

exit(0);

}

}

}

return 0;

}

Tree\* makeTree(Tree\* Root) { //Создание дерева

int key;

cout << "Конец ввода - отрицательное число\n\n";

if (Root == NULL) { // если дерево не создано

cout << "Введите ключ корня: ";

cin >> key;

Root = list(key); // установка указателя на корень

}

while (1) { //добавление элементов

cout << "\nВведите ключ: ";

cin >> key;

if (key < 0) {

break; //признак выхода (ключ < 0)

}

insertElem(Root, key);

}

return Root;

}

Tree\* list(int i) //Создание нового элемента

{

Tree\* t = new Tree[sizeof(Tree)];

t->key = i;

t->Left = t->Right = NULL;

return t;

}

Tree\* insertElem(Tree\* t, int key) //Добавление нового элемента

{

Tree\* Prev; // Prev - элемент перед текущим

Prev = t;

int find = 0; // признак поиска

while (t && !find) {

if (key == t->key) {

find = 1; //ключи должны быть уникальны

}

else {

if (key < t->key) {

t = t->Left;

}

else {

t = t->Right;

}

}

}

if (!find) { //найдено место с адресом Prev

t = list(key); //создается новый узел

if (key < Prev->key) { // и присоединяется либо

Prev->Left = t; //переход на левую ветвь,

}

else {

Prev->Right = t; // либо на правую

}

}

return t;

}

Tree\* delet(Tree\* Root, int key) //Удаление элемента по ключу

{ // Del, Prev\_Del - удаляемый элемент и его предыдущий;

// R, Prev\_R - элемент, на который заменяется удаленный, и его родитель;

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

Del = Root;

Prev\_Del = NULL;

while (Del != NULL && Del->key != key)//поиск элемента и его родителя

{

Prev\_Del = Del;

if (Del->key > key)

Del = Del->Left;

else

Del = Del->Right;

}

if (Del == NULL) // элемент не найден

{

puts("\nНет такого ключа");

return Root;

}

if (Del->Right == NULL) // поиск элемента R для замены

R = Del->Left;

else

if (Del->Left == NULL)

R = Del->Right;

else

{

Prev\_R = Del; //поиск самого правого элемента в левом поддереве

R = Del->Left;

while (R->Right != NULL)

{

Prev\_R = R;

R = R->Right;

}

if (Prev\_R == Del) // найден элемент для замены R и его родителя Prev\_R

R->Right = Del->Right;

else

{

R->Right = Del->Right;

Prev\_R->Right = R->Left;

R->Left = Prev\_R;

}

}

if (Del == Root) Root = R; //удаление корня и замена его на R

else

// поддерево R присоединяется к родителю удаляемого узла

if (Del->key < Prev\_Del->key)

Prev\_Del->Left = R; //на левую ветвь

else

Prev\_Del->Right = R; //на правую ветвь

int tmp = Del->key;

cout << "\nУдален элемент с ключом " << tmp << endl;

delete Del;

return Root;

}

Tree\* search(Tree\* n, int key) { //Поиск элемента по ключу

Tree\* rc = n;

if (rc != NULL) {

if (key < (key, n->key)) {

rc = search(n->Left, key);

}

else

if (key > (key, n->key)) {

rc = search(n->Right, key);

}

}

else

cout << "Нет такого элемента\n";

return rc;

}

int view(Tree\* t, int level) //Вывод дерева

{

if (t) {

view(t->Right, level + 1); //вывод правого поддерева

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

cout << " ";

}

int tm = t->key;

cout << tm << ' ';

view(t->Left, level + 1); //вывод левого поддерева

return 0;

}

return 1;

}

bool delAll(Tree\* t) { //Очистка дерева

if (t != NULL) {

delAll(t->Left);

delAll(t->Right);

delete t;

return true;

}

return false;

}

int getCoincidences(Tree\* t, int k, int amount) {

if (t == nullptr) { // Если дерево пустое

return 0;

}

if (t->key == k) {

amount++; // Увеличиваем счетчик, если ключ текущей вершины равен k

}

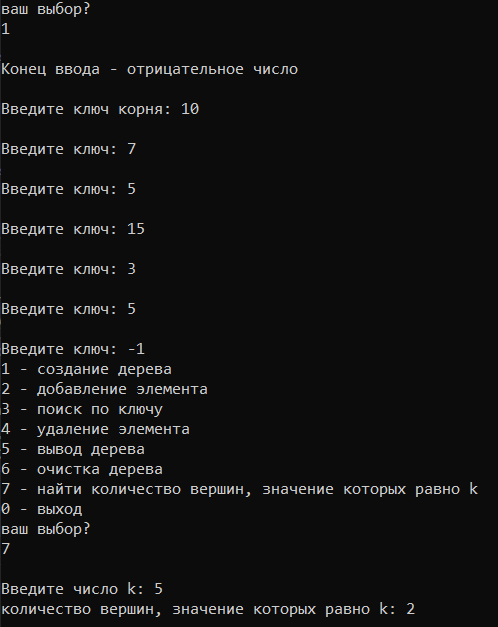
// Рекурсивно обходим левое и правое поддеревья, суммируя количество совпадений

amount += getCoincidences(t->Left, k, amount);

amount += getCoincidences(t->Right, k, amount);

return amount;

}



Вариант 12

#include <iostream>

#include <windows.h>

using namespace std;

struct Tree //дерево

{

int key; //ключ

Tree\* Left, \* Right;

};

Tree\* makeTree(Tree\* Root); //Создание дерева

Tree\* list(int i); //Создание нового элемента

Tree\* insertElem(Tree\* Root, int key); //Добавление нового элемента

Tree\* search(Tree\* n, int key); //Поиск элемента по ключу

Tree\* delet(Tree\* Root, int key); //Удаление элемента по ключу

int view(Tree\* t, int level); //Вывод дерева

bool delAll(Tree\* t); //Очистка дерева

int getEvenAmount(Tree\* t);

Tree\* Root = NULL; //указатель на корень

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int key, choice;

Tree\* rc;

for (;;)

{

cout << "1 - создание дерева\n";

cout << "2 - добавление элемента\n";

cout << "3 - поиск по ключу\n";

cout << "4 - удаление элемента\n";

cout << "5 - вывод дерева\n";

cout << "6 - очистка дерева\n";

cout << "7 - определить количество узлов с четными ключами\n";

cout << "0 - выход\n";

cout << "ваш выбор?\n";

cin >> choice;

cout << "\n";

switch (choice) {

case 1: {

Root = makeTree(Root);

break;

}

case 2: {

cout << "\nВведите ключ: ";

cin >> key;

insertElem(Root, key);

break;

}

case 3: {

cout << "\nВведите ключ: ";

cin >> key;

rc = search(Root, key);

break;

}

case 4: {

cout << "\nВведите удаляемый ключ: ";

cin >> key;

Root = delet(Root, key);

break;

}

case 5: {

if (Root->key >= 0) {

cout << "Дерево повернуто на 90 град. влево" << endl;

view(Root, 0);

}

else {

cout << "Дерево пустое\n";

break;

}

}

case 6: {

delAll(Root);

break;

}

case 7: {

cout << "Количество узлов с четным ключем: " << getEvenAmount(Root) << endl;

break;

}

case 0: {

exit(0);

}

}

}

return 0;

}

Tree\* makeTree(Tree\* Root) { //Создание дерева

int key;

cout << "Конец ввода - отрицательное число\n\n";

if (Root == NULL) { // если дерево не создано

cout << "Введите ключ корня: ";

cin >> key;

Root = list(key); // установка указателя на корень

}

while (1) { //добавление элементов

cout << "\nВведите ключ: ";

cin >> key;

if (key < 0) {

break; //признак выхода (ключ < 0)

}

insertElem(Root, key);

}

return Root;

}

Tree \* list(int i) //Создание нового элемента

{

Tree\* t = new Tree[sizeof(Tree)];

t->key = i;

t->Left = t->Right = NULL;

return t;

}

Tree\* insertElem(Tree\* t, int key) //Добавление нового элемента

{

Tree\* Prev; // Prev - элемент перед текущим

Prev = t;

int find = 0; // признак поиска

while (t && !find) {

if (key == t->key) {

find = 1; //ключи должны быть уникальны

}

else {

if (key < t->key) {

t = t->Left;

}

else {

t = t->Right;

}

}

}

if (!find) { //найдено место с адресом Prev

t = list(key); //создается новый узел

if (key < Prev->key) { // и присоединяется либо

Prev->Left = t; //переход на левую ветвь,

}

else {

Prev->Right = t; // либо на правую

}

}

return t;

}

Tree\* delet(Tree\* Root, int key) //Удаление элемента по ключу

{ // Del, Prev\_Del - удаляемый элемент и его предыдущий;

// R, Prev\_R - элемент, на который заменяется удаленный, и его родитель;

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

Del = Root;

Prev\_Del = NULL;

while (Del != NULL && Del->key != key)//поиск элемента и его родителя

{

Prev\_Del = Del;

if (Del->key > key)

Del = Del->Left;

else

Del = Del->Right;

}

if (Del == NULL) // элемент не найден

{

puts("\nНет такого ключа");

return Root;

}

if (Del->Right == NULL) // поиск элемента R для замены

R = Del->Left;

else

if (Del->Left == NULL)

R = Del->Right;

else

{

Prev\_R = Del; //поиск самого правого элемента в левом поддереве

R = Del->Left;

while (R->Right != NULL)

{

Prev\_R = R;

R = R->Right;

}

if (Prev\_R == Del) // найден элемент для замены R и его родителя Prev\_R

R->Right = Del->Right;

else

{

R->Right = Del->Right;

Prev\_R->Right = R->Left;

R->Left = Prev\_R;

}

}

if (Del == Root) Root = R; //удаление корня и замена его на R

else

// поддерево R присоединяется к родителю удаляемого узла

if (Del->key < Prev\_Del->key)

Prev\_Del->Left = R; //на левую ветвь

else

Prev\_Del->Right = R; //на правую ветвь

int tmp = Del->key;

cout << "\nУдален элемент с ключом " << tmp << endl;

delete Del;

return Root;

}

Tree\* search(Tree\* n, int key) { //Поиск элемента по ключу

Tree\* rc = n;

if (rc != NULL) {

if (key < (key, n->key)) {

rc = search(n->Left, key);

}

else

if (key > (key, n->key)) {

rc = search(n->Right, key);

}

}

else

cout << "Нет такого элемента\n";

return rc;

}

int view(Tree\* t, int level) //Вывод дерева

{

if (t) {

view(t->Right, level + 1); //вывод правого поддерева

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

cout << " ";

}

int tm = t->key;

cout << tm << ' ';

view(t->Left, level + 1); //вывод левого поддерева

return 0;

}

return 1;

}

bool delAll(Tree \* t) { //Очистка дерева

if (t != NULL) {

delAll(t->Left);

delAll(t->Right);

delete t;

return true;

}

return false;

}

int getEvenAmount(Tree\* t) {

if (t == nullptr) { //Если дерево пустое

return 0;

}

int count = 0; //Подсчет узлов с четным ключом

if (t->key % 2 == 0) {

count++;

}

count += getEvenAmount(t->Left); //Просматриваем левую ветвь

count += getEvenAmount(t->Right); //Затем правую

return count;

}

