**МИНИСТЕРСТВО НАУКИ ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**

**(национальный исследовательский университет)»**

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Структуры и алгоритмы обработки данных

Отчет по лабораторной работе № 2

**Бинарные деревья поиска**

Выполнил студент группы М3О-210Б-20  
Кильмишкин Никита Владимирович

Проверила доцент, к.т.н., Дмитриева Е.А.

Москва 2021 г.

Оглавление

[ТЗ 3](#_Toc88313665)

[Структурная схема алгоритма 4](#_Toc88313666)

[main() 4](#_Toc88313667)

[addtree() 6](#_Toc88313668)

[printTree() 7](#_Toc88313669)

[task2() 8](#_Toc88313670)

[search() 9](#_Toc88313671)

[delp() 10](#_Toc88313672)

[Листинг программы 13](#_Toc88313673)

[Результаты 29](#_Toc88313674)

[Функция поиска 29](#_Toc88313675)

[Функция добавления узла 30](#_Toc88313676)

[Функция удаления узла 31](#_Toc88313677)

[Функция обхода и находжения высоты и количества улов 34](#_Toc88313678)

[Задание №2 35](#_Toc88313679)

[Вывод 36](#_Toc88313680)

ТЗ

**Лабораторная работа «Бинарные деревья поиска»**

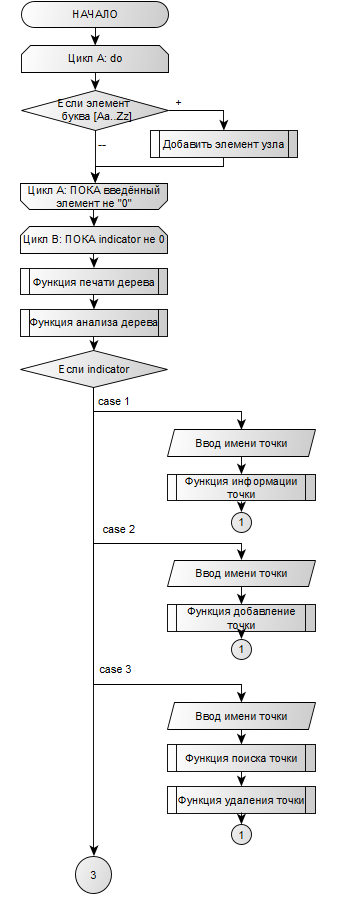
**Задание**

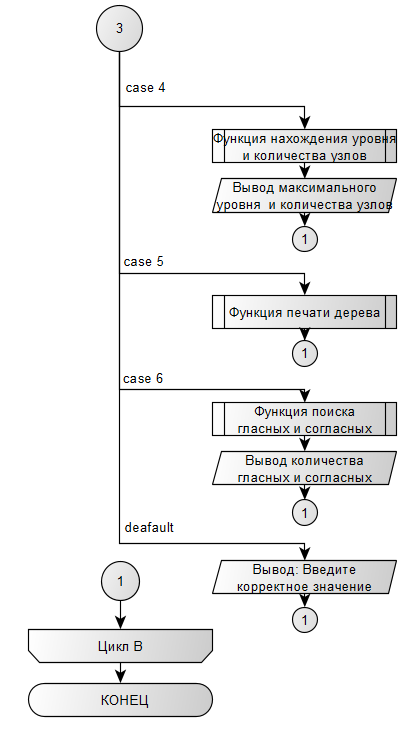
1. Реализовать функции вставки, поиска, удаления узла, обхода дерева (один из трех вариантов на выбор), вывода дерева на экран, нахождения высоты дерева и количества узлов.
2. Реализовать заданную функцию в соответствии с вариантом: *T* – тип ключей, *D* – диапазон изменения значений ключей.

|  |  |  |  |
| --- | --- | --- | --- |
| № | *T* | *D* | Функция |
| 1 | **int** | [100; 200] | Подсчет суммы длин путей от корня до каждого из узлов, содержащих четные числа |
| 2 | **char** | [а..я, А..Я] | Подсчет количества гласных в листьях |
| 3 | **int** | [0; 100] | Подсчет количества нечетных чисел в узлах, имеющих ровно два поддерева |
| 4 | **char** | [а..я, А..Я] | Подсчет количества согласных на 2 и 3 уровнях |
| 5 | **int** | [-50; 50] | Подсчет суммы четных отрицательных чисел в узлах, поддеревья которых содержат не более 4 узлов |
| 6 | **char** | [а..я, А..Я] | Подсчет количества гласных на четных уровнях |
| 7 | **int** | [0; 100] | Определить сумму четных чисел |
| 8 | **char** | [a..z, A..Z] | Подсчет количества согласных в узлах, высота поддеревьев которых одинакова |
| 9 | **char** | [a..z, A..Z] | Определить, каких букв в дереве больше - гласных или согласных |
| 10 | **char** | [а..я, А..Я] | Определить число узлов в левом и правом поддеревьях |
| 11 | **int** | [10; 90] | Определить два минимальных элемента |
| 12 | **char** | [а..я, А..Я] | Определить количество элементов дерева на каждом уровне |
| 13 | **int** | [-100; 100] | Определить, каких чисел больше - положительных или отрицательных. |
| 14 | **char** | [а..я, А..Я] | Все гласные буквы заменить на символ «\*» |
| 15 | **int** | [-50; 50] | Определить сумму элементов, кратных 5 |

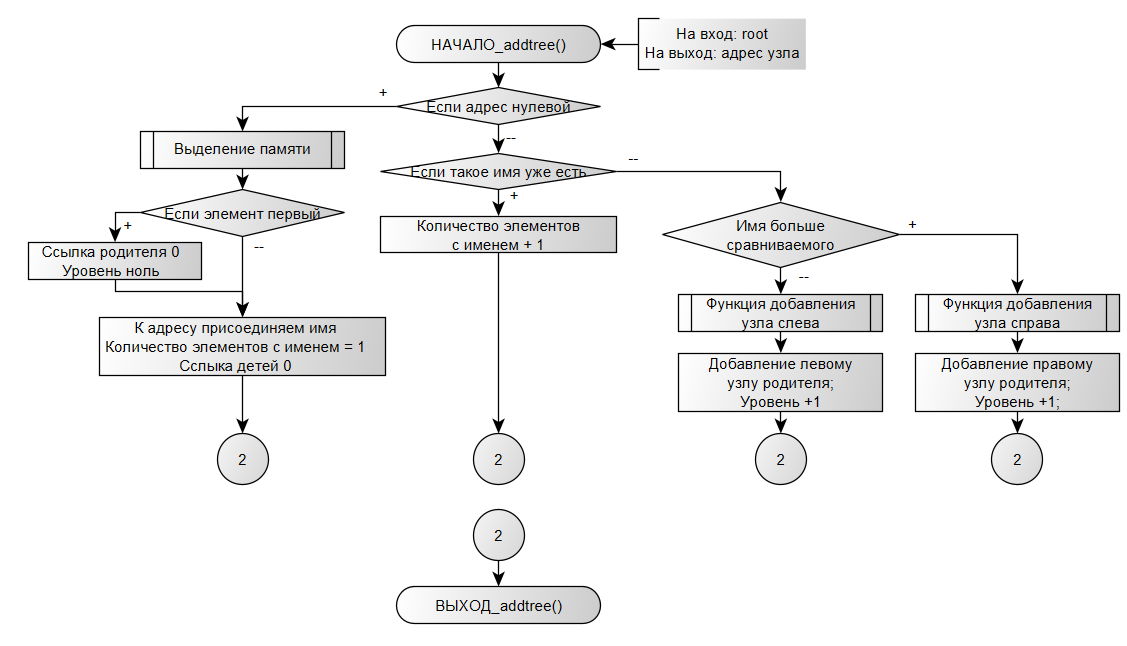
Структурная схема алгоритма

main()

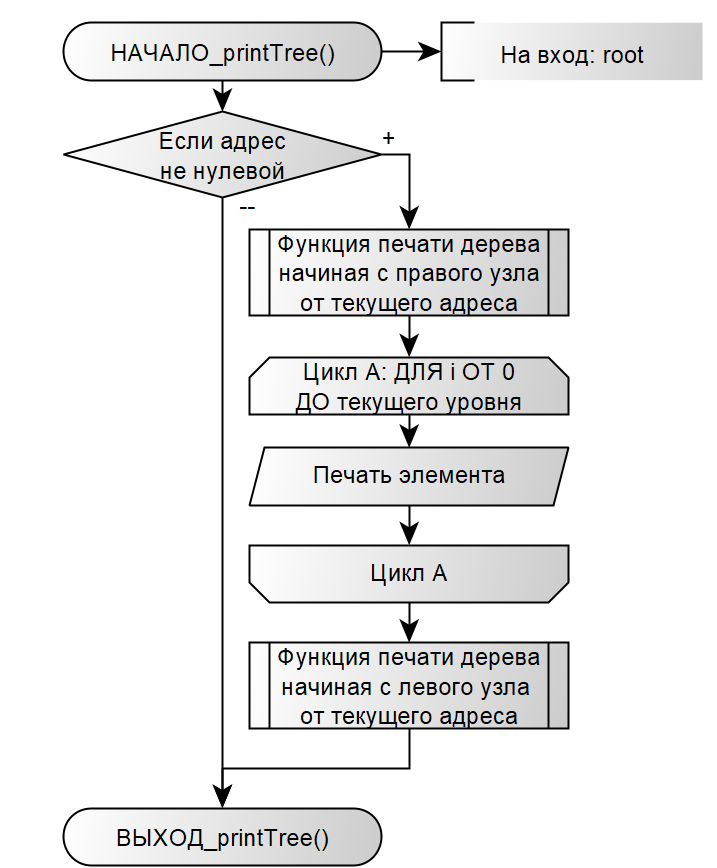




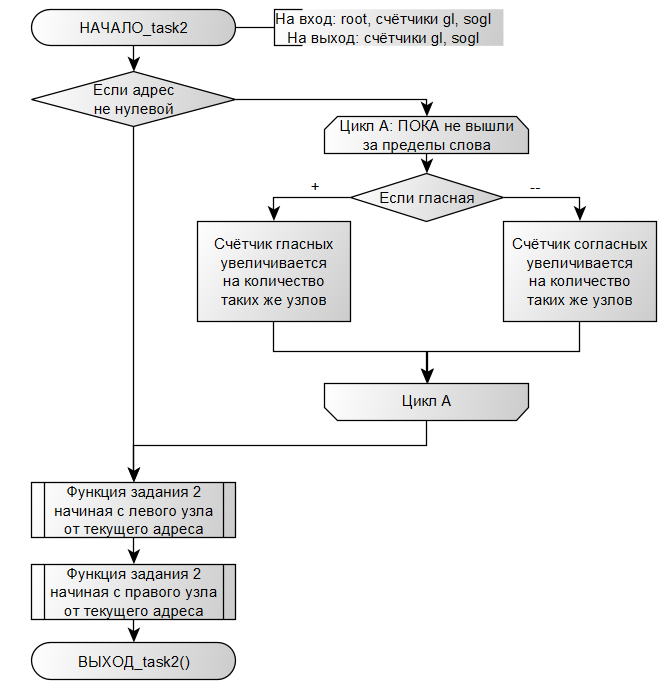
addtree()



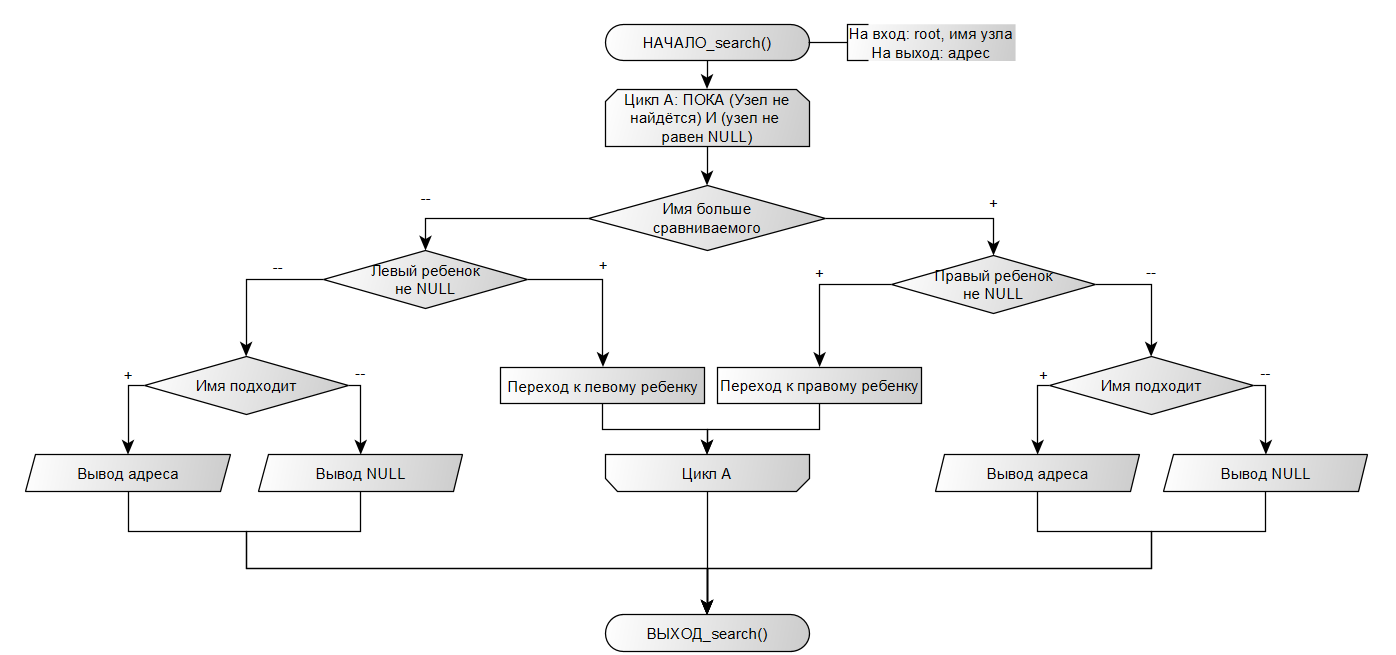
printTree()



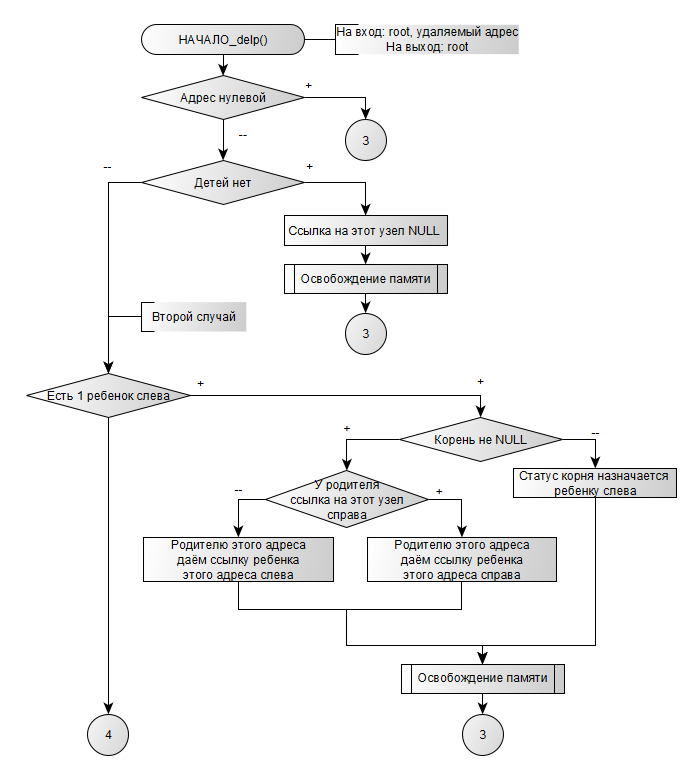
task2()

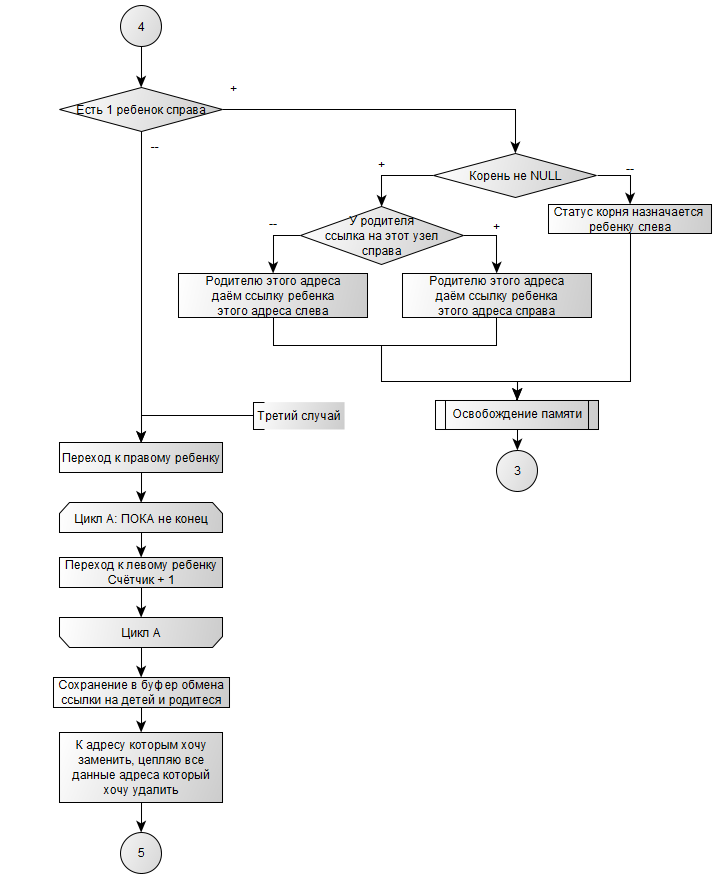


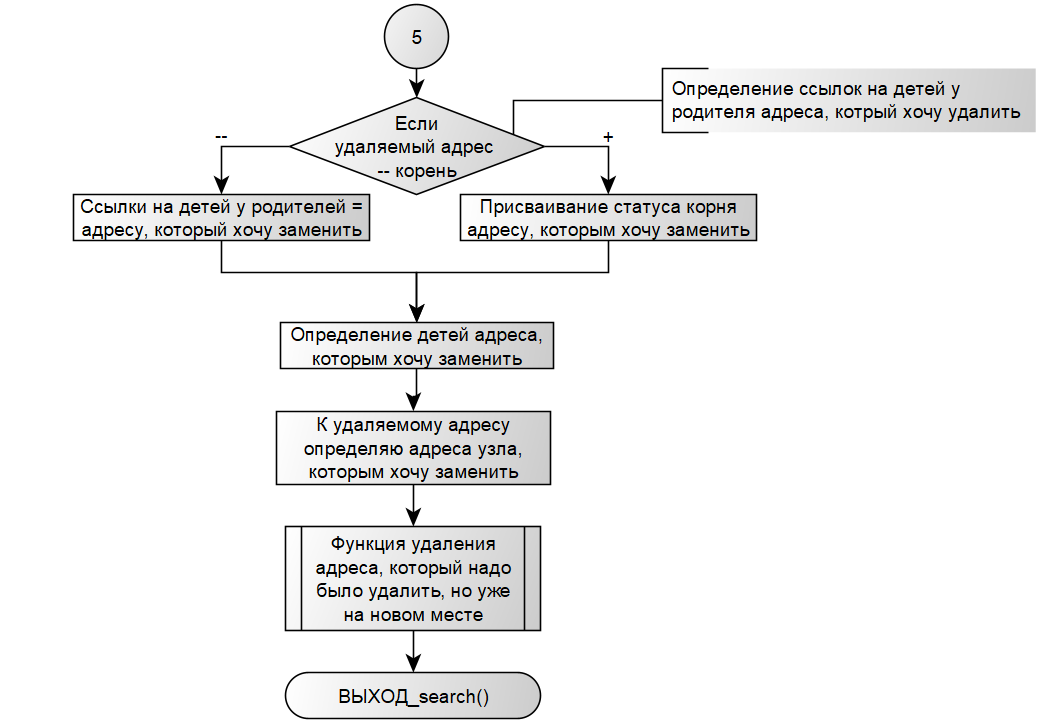
search()



delp()







Листинг программы

#include <iostream>

using namespace std;

#define MAXWORD 100

struct tnode // узел дерева

{

char\* word; // указатель на строку (слово)

int count; // число вхождений

struct tnode\* left; // левый потомок

struct tnode\* right; // правый потомок

struct tnode\* back; // родитель

int level; // уровень

};

// Функция добавления узла к дереву

struct tnode\* addtree(struct tnode\* p, char\* w)

{

//struct tnode\* temp, \* k;

int cond;

int num = 0;

if (p == NULL)

{

//p->back;

p = (struct tnode\*)malloc(sizeof(struct tnode)); // Создаем место

if (num == 0)

{

p->back = NULL;

p->level = 0;

num++;

}

p->word = \_strdup(w); // Вставляем слово

p->count = 1; // Счётчик +1

p->left = p->right = NULL; // Ссылки на концы = 0

}

else if ((cond = strcmp(w, p->word)) == 0) // Если заданое слово и слово в узле совпадут

p->count++;

else if (cond < 0) // Иначе если заданое слово больше слова в узле

{

p->left = addtree(p->left, w);

p->left->back = p;

p->left->level = p->left->back->level + 1;

}

else

{

p->right = addtree(p->right, w);

p->right->back = p;

p->right->level = p->right->back->level + 1;

}

return p;

}

// Функция удаления поддерева

void freemem(tnode\* tree) {

if (tree != NULL) {

freemem(tree->left);

freemem(tree->right);

free(tree->word);

free(tree);

}

}

// Добавление узла

void addPoint(struct tnode\* root, char\* znach)

{

if (isalpha(\*znach)) // Проверка на букву алфавита от Aa до Zz

root = addtree(root, znach);

else cout << " Введите корректное значение" << endl;

}

// Поиск узла

struct tnode\* search(struct tnode\* root, char\* key)

{

struct tnode\* point;

point = root;

cout << " Путь: " << point->word;

while ((strcmp(key, point->word) != 0) && ((point != NULL)))

{

if (strcmp(key, point->word) < 0)

{

if (point->left != NULL) point = point->left;

else if (key == point->word)

{

cout << " --> " << point->word << endl;

cout << " Я попал куда нужно" << endl;

return point;

}

else

{

cout << endl << " Штош... Не нашёл... Вот на чём остановился: " << point->word << endl;

return NULL;

}

}

else if (strcmp(key, point->word) > 0)

{

if (point->right != NULL) point = point->right;

else if (key == point->word)

{

cout << " --> " << point->word << endl;

cout << " Я попал куда нужно" << endl;

return point;

}

else

{

cout << endl << " Штош... Не нашёл... Вот на чём остановился: " << point->word << endl;

return NULL;

}

}

cout << " --> " << point->word;

}

cout << endl;

return point;

}

void datapoint(struct tnode\* root)

{

struct tnode\* point = root;

if (point)

{

cout << " Данные точки " << point->word << " :" << endl;

cout << " " << point->word << "->point\t: " << point << ", ";

if (point) cout << point->word; cout << endl;

cout << " " << point->word << "->left\t: " << point->left << ", ";

if (point->left) cout << point->left->word; cout << endl;

cout << " " << point->word << "->right\t: " << point->right << ", ";

if (point->right) cout << point->right->word; cout << endl;

cout << " " << point->word << "->back\t: " << point->back << ", ";

if (point->back) cout << point->back->word; cout << endl;

cout << " " << point->word << "->level\t: " << point->level << endl;

cout << " " << point->word << "->count\t: " << point->count << endl << endl;

}

}

// Удаление узла

void delp(struct tnode\*\* root, struct tnode\* point)

{

//struct tnode\* point = (\*root);

//datapoint(\*root);

if (point == NULL)

{

cout << " Удалить невозможно, данного узла не существует" << endl

<< " попробуйте для начала его создать" << endl;

return;

}

cout << " Удаляем: " << point->word << endl;

if ((point->left == NULL) && (point->right == NULL)) // Если точка конечная

{

if (point == point->back->left)

{

point->back->left = NULL;

free(point);

}

else

{

point->back->right = NULL;

free(point);

}

}

// Наследник в левой ветви

else if (point->right == NULL)

{

if (point->back != NULL)

// Предок в правой ветви (Есть)

if (point == point->back->right)

{

point->back->right = point->left;

point->left->back = point->back;

point->left->level = point->level;

//free(point);

}

// Предок в левой ветви (есть)

else

{

point->back->left = point->left;

point->left->back = point->back;

point->left->level = point->level;

//free(point);

}

else // Если удаляем корень

{

(\*root) = point->left;

point->left->back = NULL;

}

free(point);

}

// Наследник в правой ветви

else if (point->left == NULL)

{

if (point->back != NULL)

// Предок в правой ветви (есть)

if (point == point->back->right)

{

point->back->right = point->right;

point->right->back = point->back;

point->right->level = point->level;

//free(point);

}

// Предок в левой ветви

else

{

point->back->left = point->right;

point->right->back = point->back;

point->right->level = point->level;

//free(point);

}

else // Если удаляем корень

{

(\*root) = point->right;

point->right->back = NULL;

}

free(point);

}

else // Третий случай

{

struct tnode\* target = point;

int flag = 0; // Счётчик и индикатор спускания вниз

target = target->right; // Всегда

while (target->left != NULL) // Спуск вниз

{

target = target->left; // Самый наименьший из узлов больше удаляемого

flag++;

}

struct tnode\* pleft = target->left;

struct tnode\* pright = target->right;

struct tnode\* pback = target->back;

cout << " Третий случай" << endl;

cout << " Скопировал target" << endl << endl;

cout << " Должен равняться target" << endl;

cout << " pleft \t: " << pleft << endl;

cout << " pright\t: " << pright << endl;

cout << " pback \t: " << pback << endl << endl;

cout << " Это таргет, на эту точку буду заменять: " << endl;

datapoint(target);

cout << " Это поинт, эту точку буду удалять: " << endl;

datapoint(point);

cout << " flag: " << flag << endl << endl;

// К адресу таргет цепляю все адреса поинт

if (flag) // Цепляю данные таргет

target->right = point->right;

else

target->right = point;

target->left = point->left;

target->level = point->level;

target->back = point->back;

target->count = point->count;

cout << " target: " << target;

if (point->back) // Если не корневой

{

if (point->back->right == point) // Цепляю родителей к таргету

point->back->right = target;

else

point->back->left = target;

}

else (\*root) = target;

point->left->back = target; // Цепляю детей к таргету

if (flag)

point->right->back = target;

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

cout << " point должен совпадать с target: " << endl;

cout << " Поинт: " << endl;

datapoint(point);

cout << " Таргет: " << endl;

datapoint(target);

// к поинт цепляю все темп (то что было у таргет)

point->right = pright;

point->left = pleft;

point->back = pback;

if (flag)

pback->left = point;

else

point->back = target;

cout << endl;

cout << " point должен равняться temp: " << endl;

cout << " temp:" << endl;

cout << " pleft : " << pleft << endl;

cout << " pright: " << pright << endl;

cout << " pback: " << pback << endl << endl;

cout << " Поинт: " << endl;

datapoint(point);

cout << " Таргет: " << endl;

datapoint(target);

cout << " root: " << endl;

datapoint(\*root);

delp(root, point);

}

}

// Печать дерева

void printTree(struct tnode\* root, int level)

{

if (root)

{

printTree(root->right, level + 1); // вывод правого поддерева

for (int i = 0; i < level; i++) cout << "\t";

cout << " " << root->word << endl; // вывод корня поддерева

printTree(root->left, level + 1); // вывод левого поддерева

}

}

// Прямой обход в глубину

void preOrderTravers(tnode\* root, int& mlevel, int& count)

{

int level = mlevel;

if (root)

{

cout << root->word << " "; // Вывод узла на экран

level = root->level; // | Поиск

if (mlevel < level) mlevel = level; // | максимального уровня

count++; // Счётчик количества узлов

preOrderTravers(root->left, mlevel, count);

preOrderTravers(root->right, mlevel, count);

}

}

void BotBorder(){ cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl; }

void glsogl(char\* word, int& gl, int& sogl)

{

const char\* p = word;

while (\*p)

{

if ((\*p == 'A') || (\*p == 'E') || (\*p == 'I') || (\*p == 'O') || (\*p == 'U') || (\*p == 'Y') ||

(\*p == 'a') || (\*p == 'e') || (\*p == 'i') || (\*p == 'o') || (\*p == 'u') || (\*p == 'y'))

{

gl++;

//(\*glas) = (char\*\*)malloc(sizeof(char));

}

else

{

sogl++;

//(\*soglas) = (char\*\*)malloc(sizeof(char));

}

++p;

}

}

void task2(tnode\* root, int& gl, int& sogl)

{

if (root)

{

glsogl(root->word, gl, sogl);

task2(root->left, gl, sogl);

task2(root->right, gl, sogl);

}

}

int main()

{

setlocale(LC\_ALL, "Ru");

struct tnode\* root; // Адрес корня дерева

struct tnode\* point; // Адрес для работы

char word[MAXWORD];

char spisok[MAXWORD][MAXWORD] = { "O", "E", "P", "C", "L", "T", "J", "M", "R", "W", "F", "N", "Q", "S", "G", "0" };

//char spisok[MAXWORD][MAXWORD] = { "O", "E", "C", "L", "J", "M", "F", "N", "G", "0" };

char glas[MAXWORD][MAXWORD];

char soglas[MAXWORD][MAXWORD];

int indicator = 1; // Значение для цикла while

int count = 0; // Количество узлов

int maxlevel = 0; // Максимальный уровень

root = NULL; // Обнуление корня

// Задачи:

// Допилить случай, когда вводишь несуществующий элемент

cout << " Чтение узлов с массива..." << endl;

do {

//cout << " " << count << ": ";

//scanf\_s("%s", word, MAXWORD);

if (isalpha(\*spisok[count])) // Проверка на букву алфавита от Aa до Zz

root = addtree(root, spisok[count]);

count++;

} while (\*spisok[count] != '0'); // Условие выхода – ввод символа '0'

BotBorder();

while (indicator)

{

cout << " Дерево: " << endl;

cout << " lvl:0\t" << "lvl:1\t" << "lvl:2\t" << "lvl:3\t" << "lvl:4\t" << "lvl:5\t" << "lvl:6\t" << endl;

printTree(root, 0); cout << endl;

cout << " Как же я люблю узлы, вот они слева направо : " << endl << " " ;

preOrderTravers(root, maxlevel, count); cout << endl;

cout << " Корень: root = " << root->word << endl << endl;

cout << " Меню: " << endl;

cout << " 1: Информация точки" << endl;

cout << " 2: Добавить узел" << endl;

cout << " 3: Найти узел" << endl;

cout << " 4: Удалить узел" << endl;

cout << " 5: Обход дерева, нахождение высоты дерева и количества узлов" << endl;

cout << " 6: Вывод дерева на экран" << endl;

cout << " 7: Задание №2" << endl;

cout << " 0: ВЫХОД" << endl;

BotBorder();

cout << " Значение: "; cin >> indicator; cout << endl;

switch (indicator)

{

case 1:

{

cout << "\tИнформация точки:" << endl;

cout << " Введите точку: ";

char info[MAXWORD];

scanf\_s("%s", info, MAXWORD);

datapoint(search(root, info));

break;

}

case 2: // Добавление узла

{

cout << "\t\tДобавление узла:" << endl;

cout << " Введите значение: ";

char addp[MAXWORD];

scanf\_s("%s", addp, MAXWORD);

addPoint(root, addp);

break;

}

case 3: // Поиск

{

cout << "\t\tПоиск узла: " << endl;

cout << " Введите значение: ";

char key[MAXWORD];

scanf\_s("%s", key, MAXWORD);

datapoint(search(root, key));

break;

}

case 4: // Удаление

{

cout << "\t\tУдаление узла:" << endl;

cout << " Выберите какой узел удалить: ";

char delword[MAXWORD];

struct tnode\* delpoint; // Адрес для работы

scanf\_s("%s", delword, MAXWORD);

delpoint = search(root, delword);

delp(&root, delpoint);

break;

}

case 5: // Обход

{

cout << endl << "\t\t Прямой обход в глубину"

<< endl << "\t\t нахождение высоты дерева"

<< endl << "\t\tНахождение количества узлов" << endl;

int count = 0;

int maxlevel = 0;

preOrderTravers(root, maxlevel, count);

cout << endl << " Конец перехода" << endl;

cout << " Итого: " << endl;

cout << " Высота дерева:" << maxlevel << endl;

cout << " Количество узлов: " << count << endl;

break;

}

case 6: // Печать дерева

{

cout << " Дерево: ";

printTree(root, 0); cout << endl;

break;

}

case 7: // Задание 2

{

//char slovo[MAXWORD];

int gl = 0; // Счётчик согласных

int sogl = 0; // Счётчик гласных

task2(root, gl, sogl);

cout << " Гласные: " << gl << endl;

cout << " Согласные: " << sogl << endl;

}

case 0:

{

cout << "\t\t До новых встреч!" << endl;

break;

}

default:

{

cout << " Введите корректное значение" << endl;

break;

}

}

//system("cls");

cout << "+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++" << endl;

}

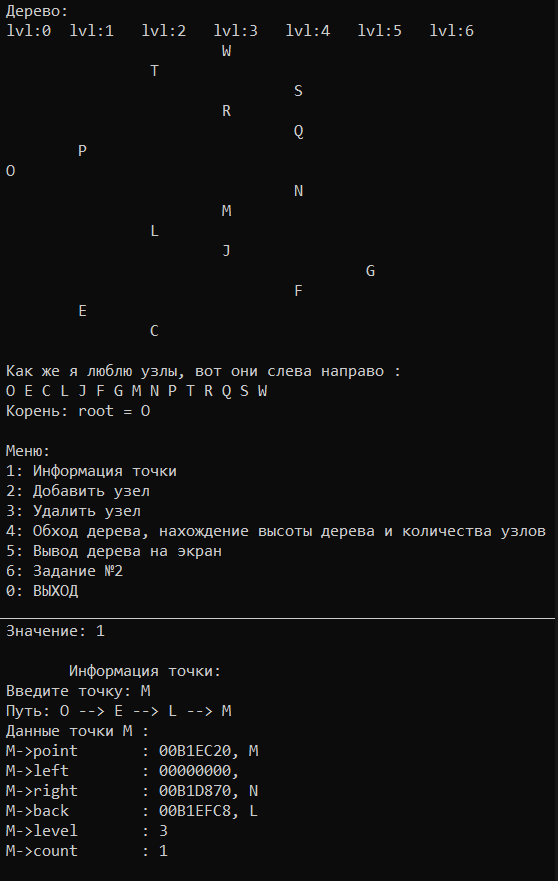
freemem(root);

return 0;

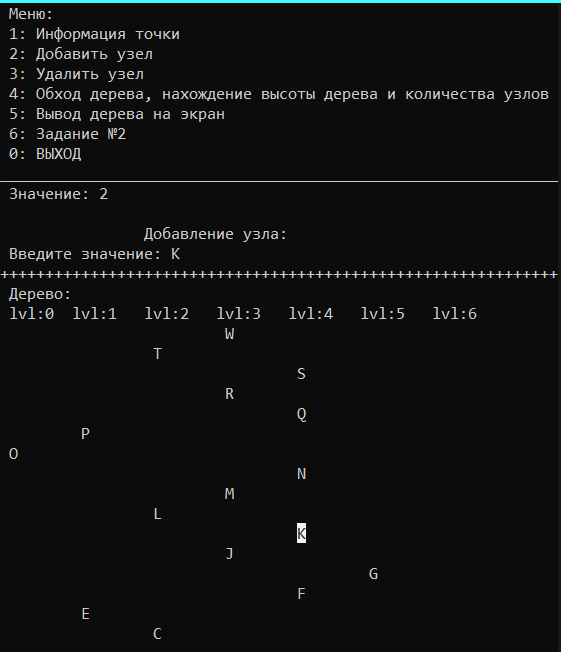
}

Результаты

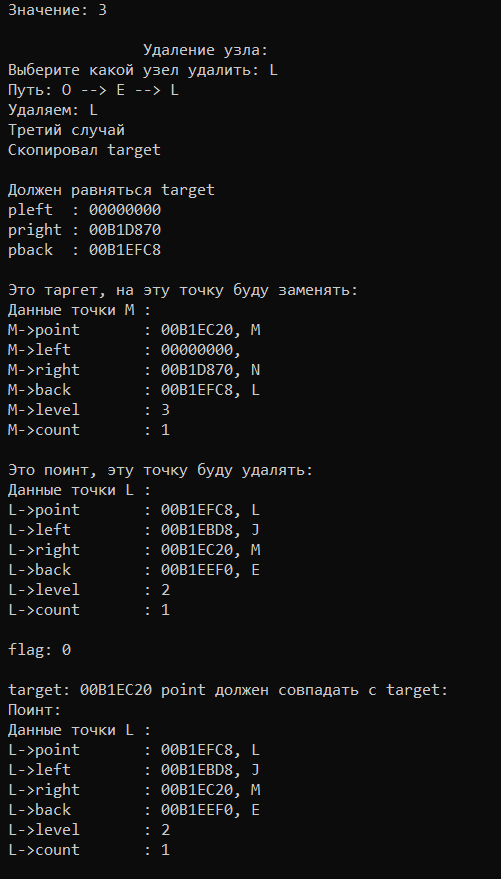
Функция поиска

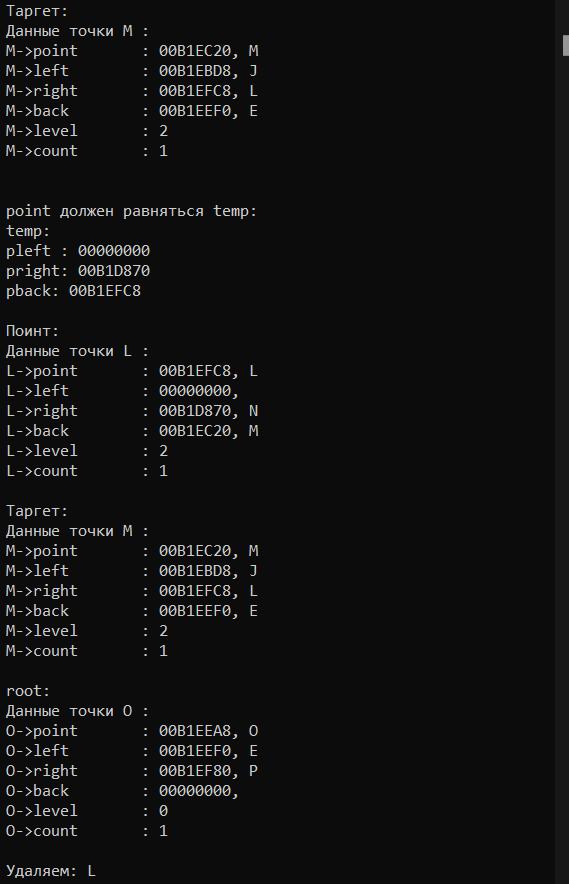
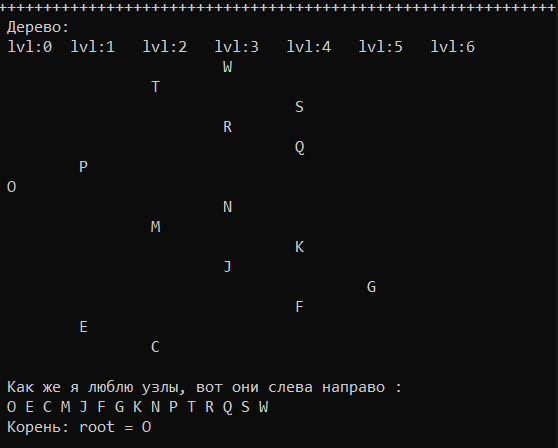


Функция добавления узла

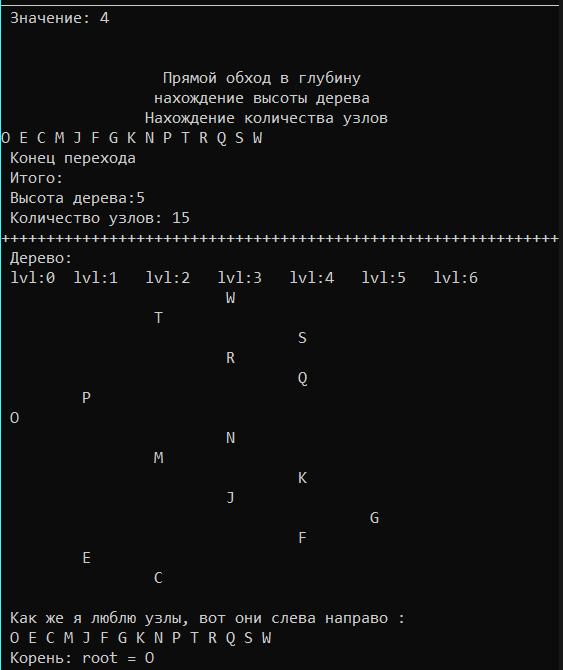


Функция удаления узла

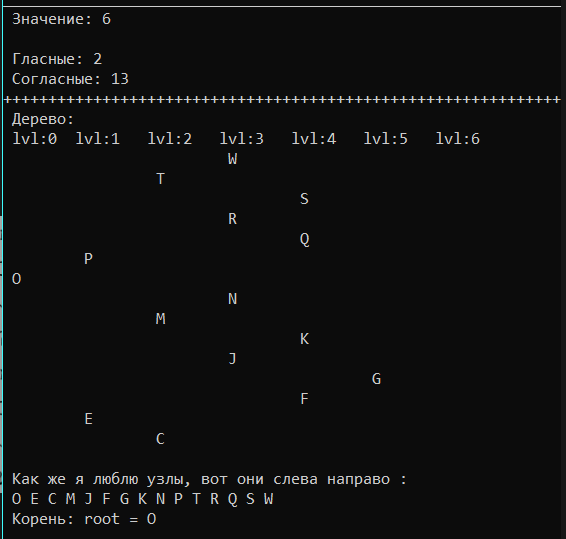


Функция обхода и находжения высоты и количества улов



Задание №2



Вывод

В лабораторной работе были реализованы функции вставки, поиска, удаления узла, обхода дерева (один из трех вариантов на выбор), вывода дерева на экран, нахождения высоты дерева и количества узлов.

Также были реализованы заданную функцию в соответствии с вариантом с типом ключей char, с диапазоном изменения значений ключей [Aa...Zz].