Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа № 11

По дисциплине «Основы алгоритмизации и программирования»

На тему «**Бинарные деревья** »

Выполнил:

Федорович Вадим

Студент 1 курса 8 группы

Преподаватель: асс. Андронова М.В.

Минск, 2024

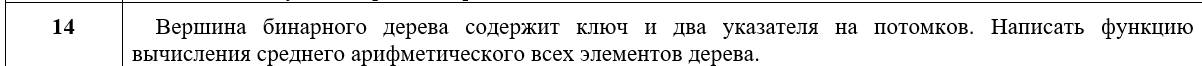
ВАЖНО!!!

Уважаемая Мария Васильевна, в силе неподвластных мне обстоятельств, таких как поломка компьютера, я не мог закинуть лабу до предыдущей среды.

Надеясь на вашу лояльность, прошу вас засчитать мне допы, если я не успел закинуть вовремя☺

Заранее, спасибо

Основа:



#include <iostream>

using namespace std;

struct Tree //дерево

{

int key; //ключ

char text[5]; //текст - не более 4 букв

Tree\* Left, \* Right;

};

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

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

Tree\* insertElem(Tree\* Root, int key, char\* s); //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

int count(Tree\* t, char letter); //Подсчет количества слов

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

int calculate(Tree\* t, int level, int& sum, int& count); //вычисление среднего арифметического всех элементов дерева

int c = 0; //количество слов

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

void main()

{

setlocale(0, "");

int key, choice, n;

Tree\* rc; char s[5], letter;

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 - подсчет количества букв\n";

cout << "7 - очистка дерева\n";

cout << "8 - выход\n";

cout << "9 - вычисление среднего арифметического всех элементов дерева\n";

cout << "ваш выбор?\n";

cin >> choice;

cout << "\n";

switch (choice)

{

case 1: Root = makeTree(Root); break;

case 2: cout << "\nВведите ключ: ";

cin >> key;

cout << "Введите слово: ";

cin >> s;

insertElem(Root, key, s); break;

case 3: cout << "\nВведите ключ: ";

cin >> key;

rc = search(Root, key);

cout << "Найденное слово= ";

puts(rc->text); 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: cout << "\nВведите букву: ";

cin >> letter;

n = count(Root, letter);

cout << "Количество слов, начинающихся с буквы " << letter;

cout << " равно " << n << endl; break;

case 7: delAll(Root); break;

case 8: exit(0);

case 9:

int sum = 0, digit = 0;

calculate(Root, 0, sum, digit);

cout <<"Среднее арифметическое значений узлов дерева :" << sum / digit<<endl;

}

}

}

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

{

int key; char s[5];

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

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

{

cout << "Введите ключ корня: ";

cin >> key;

cout << "Введите слово корня: ";

cin >> s;

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

}

while (1) //добавление элементов

{

cout << "\nВведите ключ: ";

cin >> key;

if (key < 0) break; //признак выхода (ключ < 0)

cout << "Введите слово: ";

cin >> s;

insertElem(Root, key, s);

}

return Root;

}

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

{

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

t->key = i;

for (i = 0; i < 5; i++)

\*((t->text) + i) = \*(s + i);

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

return t;

}

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

{

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

int find = 0; // признак поиска

while (t && !find)

{

Prev = t;

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, s); //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 count(Tree\* t, char letter) //Подсчет количества слов

{

if (t)

{

count(t->Right, letter);

if (\*(t->text) == letter)

c++;

count(t->Left, letter);

}

return c;

}

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 << ' ';

puts(t->text);

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 calculate(Tree\* t, int level, int &sum, int& digit) // Вывод дерева

{

if (t)

{

calculate(t->Right, level + 1, sum, digit); // вывод правого поддерева

sum += t->key; // изменено: разыменование указателя sum

digit ++; // изменено: разыменование указателя digit

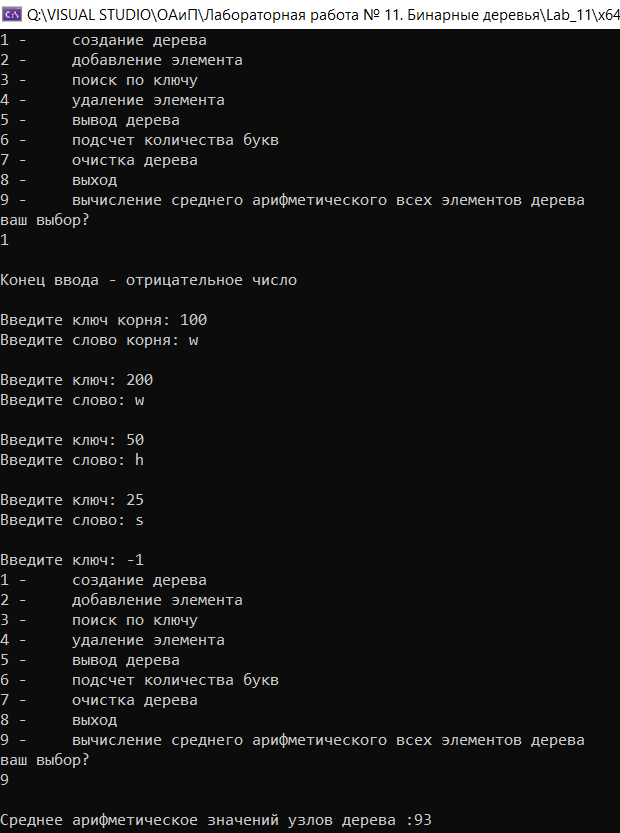
calculate(t->Left, level + 1, sum, digit); // вывод левого поддерева

return 0;

}

return 1;

}



Допы:



#include <iostream>

using namespace std;

struct Tree //дерево

{

int key; //ключ

char text[5]; //текст - не более 4 букв

Tree\* Left, \* Right;

};

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

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

Tree\* insertElem(Tree\* Root, int key, char\* s); //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

int count(Tree\* t, char letter); //Подсчет количества слов

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

int count\_Right\_side(Tree\* t); //Подсчет количества слов

int c = 0; //количество слов

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

void main()

{

setlocale(0, "Russian");

int key, choice, n;

Tree\* rc{}; char s[5], letter;

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 - подсчет количества букв\n";

cout << "7 - очистка дерева\n";

cout << "8 - выход\n";

cout << "8 - Поиск кол-ва эл правой стороны\n";

cout << "ваш выбор?\n";

cin >> choice;

cout << "\n";

switch (choice)

{

case 1: Root = makeTree(Root); break;

case 2: cout << "\nВведите ключ: ";

cin >> key;

cout << "Введите слово: ";

cin >> s;

insertElem(Root, key, s); break;

case 3: cout << "\nВведите ключ: ";

cin >> key;

rc = search(Root, key);

cout << "Найденное слово= ";

puts(rc->text); 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: cout << "\nВведите букву: ";

cin >> letter;

n = count(Root, letter);

cout << "Количество слов, начинающихся с буквы " << letter;

cout << " равно " << n << endl; break;

case 7: delAll(Root); break;

case 8: exit(0);

case 9: Tree \* t\_Right = Root->Right;

int s = count\_Right\_side(t\_Right); cout <<"Кол-во элементов в правом под-дереве :"<< s<<endl; break;

}

}

}

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

{

int key; char s[5];

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

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

{

cout << "Введите ключ корня: ";

cin >> key;

cout << "Введите слово корня: ";

cin >> s;

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

}

while (1) //добавление элементов

{

cout << "\nВведите ключ: ";

cin >> key;

if (key < 0) break; //признак выхода (ключ < 0)

cout << "Введите слово: ";

cin >> s;

insertElem(Root, key, s);

}

return Root;

}

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

{

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

t->key = i;

for (i = 0; i < 5; i++)

\*((t->text) + i) = \*(s + i);

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

return t;

}

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

{

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

int find = 0; // признак поиска

while (t && !find)

{

Prev = t;

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, s); //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 count(Tree\* t, char letter) //Подсчет количества слов

{

if (t)

{

count(t->Right, letter);

if (\*(t->text) == letter)

c++;

count(t->Left, letter);

}

return c;

}

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 << ' ';

puts(t->text);

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 count\_Right\_side(Tree\* t) //Подсчет количества слов

{

if (t)

{

count\_Right\_side(t->Right);

if (\*(t->text) != NULL)

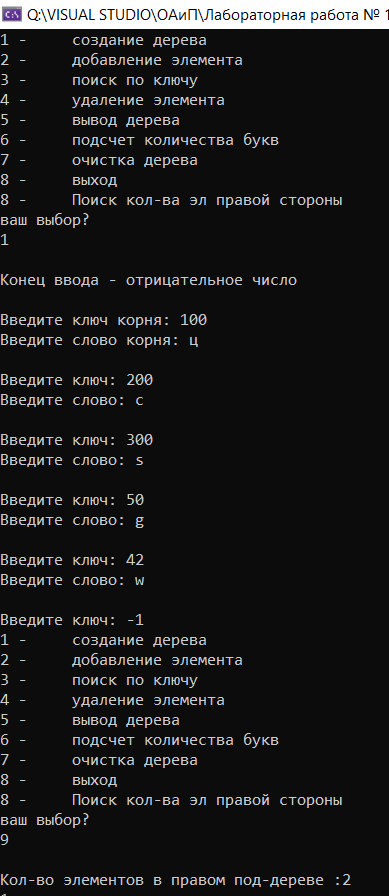
c++;

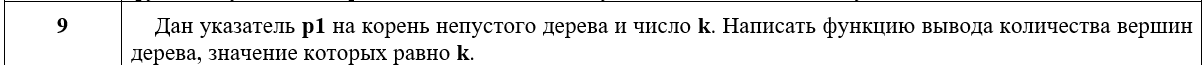
count\_Right\_side(t->Left);

}

return c;

}





#include <iostream>

using namespace std;

struct Tree //дерево

{

int key; //ключ

char text[5]; //текст - не более 4 букв

Tree\* Left, \* Right;

};

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

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

Tree\* insertElem(Tree\* Root, int key, char\* s); //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

int count(Tree\* t, char letter); //Подсчет количества слов

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

int VYVOD\_IF\_K(Tree\* t, int k, int& counter); //вычисление среднего арифметического всех элементов дерева

int c = 0; //количество слов

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

void main()

{

setlocale(0, "");

int key, choice, n;

Tree\* rc; char s[5], letter;

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 - подсчет количества букв\n";

cout << "7 - очистка дерева\n";

cout << "8 - выход\n";

cout << "9 - вычисление среднего арифметического всех элементов дерева\n";

cout << "ваш выбор?\n";

cin >> choice;

cout << "\n";

switch (choice)

{

case 1: Root = makeTree(Root); break;

case 2: cout << "\nВведите ключ: ";

cin >> key;

cout << "Введите слово: ";

cin >> s;

insertElem(Root, key, s); break;

case 3: cout << "\nВведите ключ: ";

cin >> key;

rc = search(Root, key);

cout << "Найденное слово= ";

puts(rc->text); 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: cout << "\nВведите букву: ";

cin >> letter;

n = count(Root, letter);

cout << "Количество слов, начинающихся с буквы " << letter;

cout << " равно " << n << endl; break;

case 7: delAll(Root); break;

case 8: exit(0);

case 9:

int k = 0, counter = 0;

cout << "Введите число k: ";

cin >> k;

VYVOD\_IF\_K(Root,k,counter);

cout << "Количество вершин дерева, значение которых равно k: "<< counter << endl;

}

}

}

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

{

int key; char s[5];

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

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

{

cout << "Введите ключ корня: ";

cin >> key;

cout << "Введите слово корня: ";

cin >> s;

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

}

while (1) //добавление элементов

{

cout << "\nВведите ключ: ";

cin >> key;

if (key < 0) break; //признак выхода (ключ < 0)

cout << "Введите слово: ";

cin >> s;

insertElem(Root, key, s);

}

return Root;

}

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

{

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

t->key = i;

for (i = 0; i < 5; i++)

\*((t->text) + i) = \*(s + i);

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

return t;

}

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

{

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

int find = 0; // признак поиска

while (t && !find)

{

Prev = t;

if (key < t->key)

t = t->Left;

else

t = t->Right;

}

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

{

t = list(key, s); //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 count(Tree\* t, char letter) //Подсчет количества слов

{

if (t)

{

count(t->Right, letter);

if (\*(t->text) == letter)

c++;

count(t->Left, letter);

}

return c;

}

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 << ' ';

puts(t->text);

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 VYVOD\_IF\_K(Tree\* t, int k, int &counter) // Вывод дерева

{

if (t)

{

VYVOD\_IF\_K(t->Right, k, counter); // вывод правого поддерева

if (t->key == k) {

cout << t->text<<endl;

cout << t->key<<endl;

cout << endl;

counter++;

}

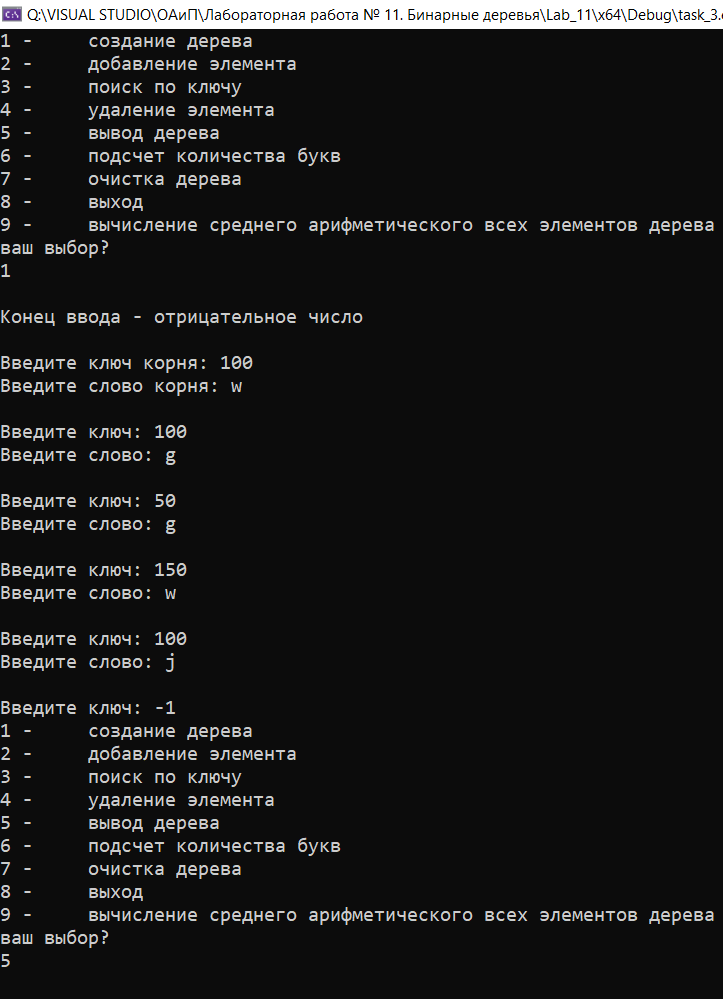
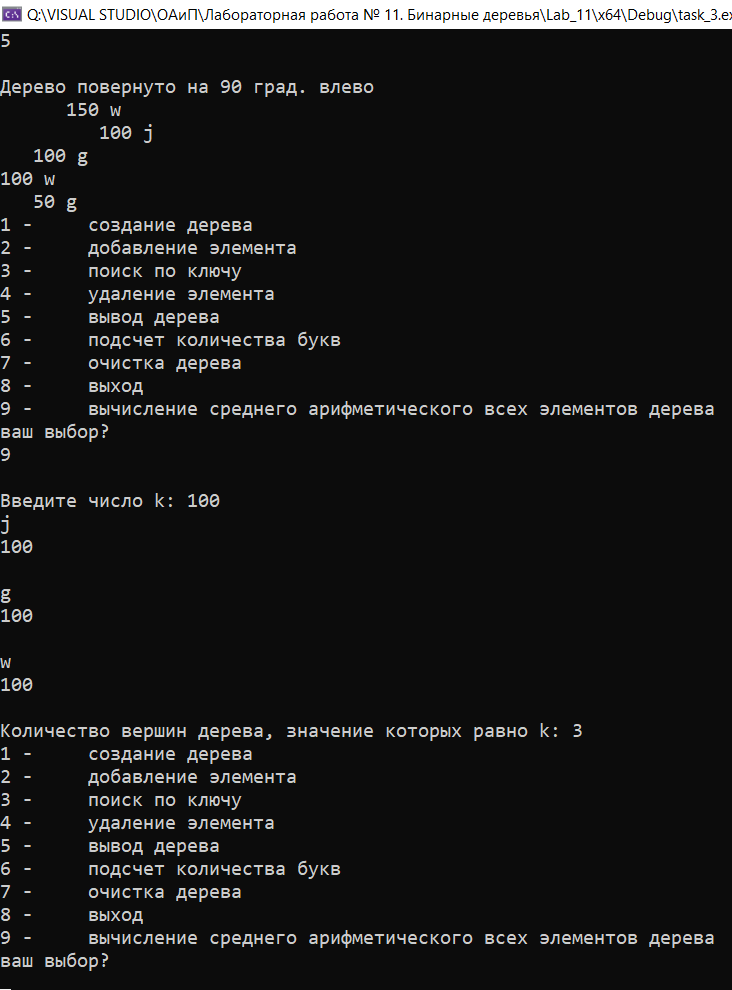
VYVOD\_IF\_K(t->Left, k, counter); // вывод левого поддерева

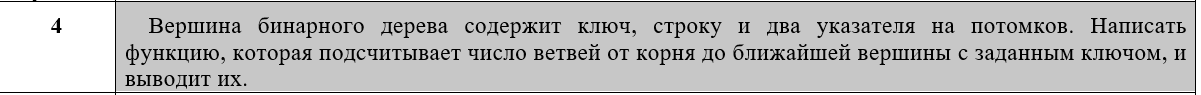
return 0;

}

return 1;

}



#include <iostream>

using namespace std;

struct Tree //дерево

{

int key; //ключ

char text[5]; //текст - не более 4 букв

Tree\* Left, \* Right;

};

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

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

Tree\* insertElem(Tree\* Root, int key, char\* s); //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

int count(Tree\* t, char letter); //Подсчет количества слов

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

int countBranchesToKRecursive(Tree\* t, int k);

int countBranchesToK(Tree\* t, int k);

int c = 0; //количество слов

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

void main()

{

setlocale(0, "");

int key, choice, n;

Tree\* rc; char s[5], letter;

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 - подсчет количества букв\n";

cout << "7 - очистка дерева\n";

cout << "8 - выход\n";

cout << "9 - вычисление среднего арифметического всех элементов дерева\n";

cout << "10 - подсчет ветвей к заданной вершине\n"; // Новый пункт меню

cout << "ваш выбор?\n";

cin >> choice;

cout << "\n";

switch (choice)

{

case 1: Root = makeTree(Root); break;

case 2: cout << "\nВведите ключ: ";

cin >> key;

cout << "Введите слово: ";

cin >> s;

insertElem(Root, key, s); break;

case 3: cout << "\nВведите ключ: ";

cin >> key;

rc = search(Root, key);

cout << "Найденное слово= ";

puts(rc->text); 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: cout << "\nВведите букву: ";

cin >> letter;

n = count(Root, letter);

cout << "Количество слов, начинающихся с буквы " << letter;

cout << " равно " << n << endl; break;

case 7: delAll(Root); break;

case 8: exit(0);

case 10: // Новый пункт меню для подсчета ветвей к заданной вершине

int k\_branches;

cout << "Введите ключ вершины, для которой нужно подсчитать ветви: ";

cin >> key;

k\_branches = countBranchesToK(Root, key);

cout << "Количество ветвей к вершине с ключом " << key << " равно " << k\_branches << endl;

break;

}

}

}

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

{

int key; char s[5];

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

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

{

cout << "Введите ключ корня: ";

cin >> key;

cout << "Введите слово корня: ";

cin >> s;

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

}

while (1) //добавление элементов

{

cout << "\nВведите ключ: ";

cin >> key;

if (key < 0) break; //признак выхода (ключ < 0)

cout << "Введите слово: ";

cin >> s;

insertElem(Root, key, s);

}

return Root;

}

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

{

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

t->key = i;

for (i = 0; i < 5; i++)

\*((t->text) + i) = \*(s + i);

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

return t;

}

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

{

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

int find = 0; // признак поиска

while (t && !find)

{

Prev = t;

if (key < t->key)

t = t->Left;

else

t = t->Right;

}

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

{

t = list(key, s); //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 count(Tree\* t, char letter) //Подсчет количества слов

{

if (t)

{

count(t->Right, letter);

if (\*(t->text) == letter)

c++;

count(t->Left, letter);

}

return c;

}

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 << ' ';

puts(t->text);

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 countBranchesToK(Tree\* t, int k)

{

if (t == nullptr) {

cout << "Дерево пустое" << endl;

return 0;

}

return countBranchesToKRecursive(t, k);

}

int countBranchesToKRecursive(Tree\* t, int k)

{

if (t == nullptr)

return 0;

if (t->key == k) {

cout << "Ключ найден: " << k << endl;

return 1;

}

if (k < t->key) {

cout << t->key << " -> ";

return 1 + countBranchesToKRecursive(t->Left, k);

}

else {

cout << t->key << " -> ";

return 1 + countBranchesToKRecursive(t->Right, k);

}

}

