# Учреждение образования

# «Белорусский государственный технологический университет»

Кафедра информационных систем и технологий

Отчёт по лабораторной работе №12

Вариант 2

Выполнил:

студент 1 курса 2 группы Самцевич Алексей

Минск 2020

Задание №1: Добавить к проекту  функции смешанного и нисходящего обхода дерева с выводом  на консоль, проверки сбалансированности дерева и функцию в соответствии с вариантом из таблицы, представленной в лабораторной работе № 11, изменив ее так, чтобы функция соответствовала проекту данной лабораторной работы.

Код:

Главный файл:

//

// main.cpp

// laba\_12.1

//

// Created by Alex Samtsevich on 08.06.2020.

// Copyright © 2020 Alex Samtsevich. All rights reserved.

//

#include <iostream>

#include <algorithm>

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

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

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

int valueElem(Tree\* t);

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

void LNR(Tree\* t);

void LRN(Tree\* t);

void balance(Tree\* t);

int c = 0; //количество слов

Tree\* Root = NULL; //указатель корня

int 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 - Кол-во листьев(правые дочерние вершины)\n";

cout << "9 - Смешанный обход \n";

cout << "10 - Нисходяий обход \n";

cout << "11 - Проверка сбалансированности \n";

cout << "12 - 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;

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: cout << "Кол-во элементов правого поддерева " << valueElem(Root) << endl; break;

case 9: LNR(Root); break;

case 10: LRN(Root); break;

case 11: balance(Root); break;

case 12: exit(0);

}

}

}

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 DOBAVUL NULL

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;

}

void 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

}

}

void delAll(Tree\* t) //Очистка дерева

{

if (t != NULL)

{

delAll(t->Left);

delAll(t->Right);

delete t;

}

}

int value(Tree\* t) {

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

return 1;

int sum = 0;

if (t->Left != NULL)

sum += value(t->Left);

if (t->Right != NULL)

sum += value(t->Right);

return sum;

}

int valueElem(Tree\* t) {

if (t->Right == NULL) {

if (t->Left == NULL)

return 1;

return 0;

}

return value(t->Right);

}

void LNR(Tree\* t) {

if (t != NULL)

{

LNR(t->Left);

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

LNR(t->Right);

}

}

void LRN(Tree\* t) {

if (t != NULL)

{

LRN(t->Left);

LRN(t->Right);

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

}

}

int height(Tree\* t) {//вспомогательная функция

if (t == NULL)

return 0;

int a = height(t->Left);

int b = height(t->Right);

return 1 + max(a, b);

}

bool isBalace(Tree\* t) {

if (t == NULL)

return true;

return (abs(height(t->Left) - height(t->Right)) <= 1) && isBalace(t->Left) && isBalace(t->Right);

}

void balance(Tree\* t) {

if (isBalace(t))

cout << "bal";

else

cout << "not bal";

}

Скриншот:



