Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Пермский национальный исследовательский политехнический университет» (ПНИПУ)

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

**Лабораторная работа по Деревьям**

**по дисциплине**

**«Основы алгоритмизации и программирования»**

**семестр** 2

Выполнил студент гр. ИВТ-21-1б

Балдин Алексей Игоревич

Проверил:

Яруллин Денис Владимирович

\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

(оценка) (подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

г. Пермь - 2022

**Цель**

Получить практические навыки работы с бинарными деревьями.

**Постановка задачи**

1. Сформировать идеально сбалансированное бинарное дерево, тип информационного поля указан в варианте.

2. Распечатать полученное дерево.

3. Выполнить обработку дерева в соответствии с заданием, вывести полученный результат.

4. Преобразовать идеально сбалансированное дерево в дерево поиска.

5. Распечатать полученное дерево.

**Исходные данные**

Вариант 5:

### Тип информационного поля char. Найти высоту дерева.

Переменные - l (левый лист), r (правый лист), tabs – количество пробелов между корнями.

Функция Node\_add – добавляет в дерево корень или листок (в зависимости от ситуации).

Функция Node\_Height – высчитывает высоту дерева, Функция Print – выводит дерево после изменений.

Реализованы методы добавления в дерево, проверяется наличие ветки, если нет – создать, потомки нулевые, следующие добавляются по аналогии с проверкой и на > (правое поддерево) или < (левое поддерево) и рекурсивно вызывается функция. Печать осуществляется также при помощи рекурсии с использованием добавления и удаления числа пробелов. Прямой обход осуществлен рекурсивно с левого поддерева – корня – правого поддерева. Функция проверка на пустоту просто проверяет, есть ли хоть одна ветка. Возвращает сообщение пустое ли дерево. Также реализован метод очистки динамической памяти.

Код программы:

#include <iostream>

#include <conio.h>

#include <iostream>

using namespace std;

struct Node {

Node\* l, \* r;

char x;

};

void Node\_Add(Node\*& p, char x);

int Node\_Height(const Node\* p);

int tabs = 0;

void print(Node\* aBranch)

{

if (!aBranch) return; //Если ветки не существует - выходим. Выводить нечего

tabs += 5; //Иначе увеличим счетчик рекурсивно вызванных процедур

//Который будет считать нам отступы для красивого вывода

print(aBranch->l); //Выведем ветку и ее подветки слева

for (int i = 0; i < tabs; i++) cout << " "; //Потом отступы

cout << aBranch->x << endl; //Данные этой ветки

print(aBranch->r);//И ветки, что справа

tabs -= 5; //После уменьшим кол-во отступов

return;

}

void is\_Empty(Node\*& aBranch)

{

if (!aBranch)

{

cout << "Дерево пустое...";

}

else

{

cout << "Дерево не пустое...";

}

}

void FreeTree(Node\* aBranch)

{

if (!aBranch) return;

FreeTree(aBranch->l);

FreeTree(aBranch->r);

delete aBranch;

return;

}

Node\* del\_elem(Node\*& aBranch, char a) {

if (aBranch == NULL)

return aBranch;

if (a == aBranch->x) {

Node\* tmp;

if (aBranch->r == NULL)

tmp = aBranch->l;

else {

Node\* ptr = aBranch->r;

if (ptr->l == NULL)

{

ptr->r = aBranch->l;

tmp = ptr;

}

else {

Node\* pmin = ptr->l;

while (pmin->l != NULL) {

ptr = pmin;

pmin = ptr->l;

}

ptr->l = pmin->r;

pmin->l = aBranch->l;

pmin->r = aBranch->r;

tmp = pmin;

}

}

delete aBranch;

return tmp;

}

else if (a < aBranch->x)

aBranch->l = del\_elem(aBranch->l, a);

else

aBranch->r = del\_elem(aBranch->r, a);

return aBranch;

}

void pr\_obh(Node\*& aBranch)

{

if (NULL == aBranch) return; //Если дерева нет, выходим

pr\_obh(aBranch->l); //Обошли левое поддерево

cout << aBranch->x << endl; //Посетили узел

pr\_obh(aBranch->r); //Обошли правое поддерево

}

//вставка

void Node\_Add(Node\*& p, char x)

{

if (p == NULL) {

p = new (std::nothrow) Node();

if (p != NULL) {

p->l = p->r = NULL;

p->x = x;

}

}

else if (x < p->x)

Node\_Add(p->l, x);

else

Node\_Add(p->r, x);

}

//высота дерева

int Node\_Height(const Node\* p) {

int l, r, h = 0;

if (p != NULL) {

l = Node\_Height(p->l);

r = Node\_Height(p->r);

h = ((l > r) ? l : r) + 1;

}

return h;

}

void add\_elem(char aData, Node\*& aBranch)

{

if (!aBranch)

{

aBranch = new Node;

aBranch->x = aData;

aBranch->l = 0;

aBranch->r = 0;

return;

}

else

{

if (aData < aBranch->x) {

add\_elem(aData, aBranch->l);

}

else if (aData > aBranch->x) {

add\_elem(aData, aBranch->r);

}

}

}

int main(void) {

Node\* tr = NULL;

setlocale(LC\_ALL, "Russian"); // Моё

int n;

char x;

cout << "Сколько узлов будет в дереве ? (Введите кол-во узлов (min : 2))" << endl;

for (int menu = 0; menu == 0;)

{

cin >> n;

if (n > 1)

{

menu++;

}

else

{

cout << "Введено неправильное кол-во узлов. Попробуйте еще." << endl;

}

}

is\_Empty(tr);

cout << endl << endl;

char\* s = new char[n];

int count = 0;

for (; count < n; count++)

{

cout << "Введите элемент " << count + 1 << " : ";

cin >> s[count];

cout << endl;

Node\_Add(tr, s[count]);

}

is\_Empty(tr);

cout << endl;

cout << "Вывод бинарного дерева: " << endl;

print(tr);

cout << endl;

cout << "Прямой обход бинарного дерева: " << endl;

pr\_obh(tr);

cout << endl;

cout << "Добавим новый элемент в бинарное дерево:" << endl;

cout << "Введите новый элемент: ";

cin >> x;

add\_elem(x, tr);

cout << "Вывод бинарного дерева: " << endl;

print(tr);

cout << endl;

cout << "Удалим элемент из бинарного дерева:" << endl;

cout << "Введите элемент: ";

cin >> x;

del\_elem(tr, x);

cout << "Вывод бинарного дерева: " << endl;

print(tr);

cout << endl;

cout << "-------------------------------" << endl;

cout << "Высота дерева равна: " << Node\_Height(tr) << endl << endl;

FreeTree(tr);

cout << endl;

cout << "Вся динамическая память очищена..." << endl;

cout << "-------------------------------" << endl;

cout << endl << endl;

return 0;

}

Результат программы:

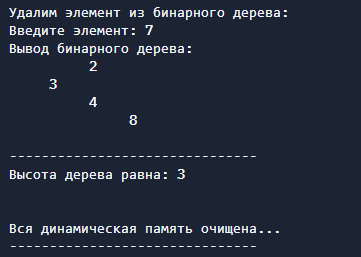
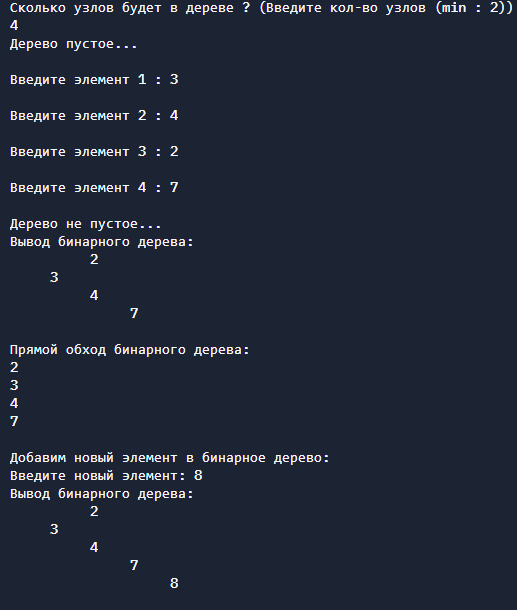


Диаграмма к задаче:

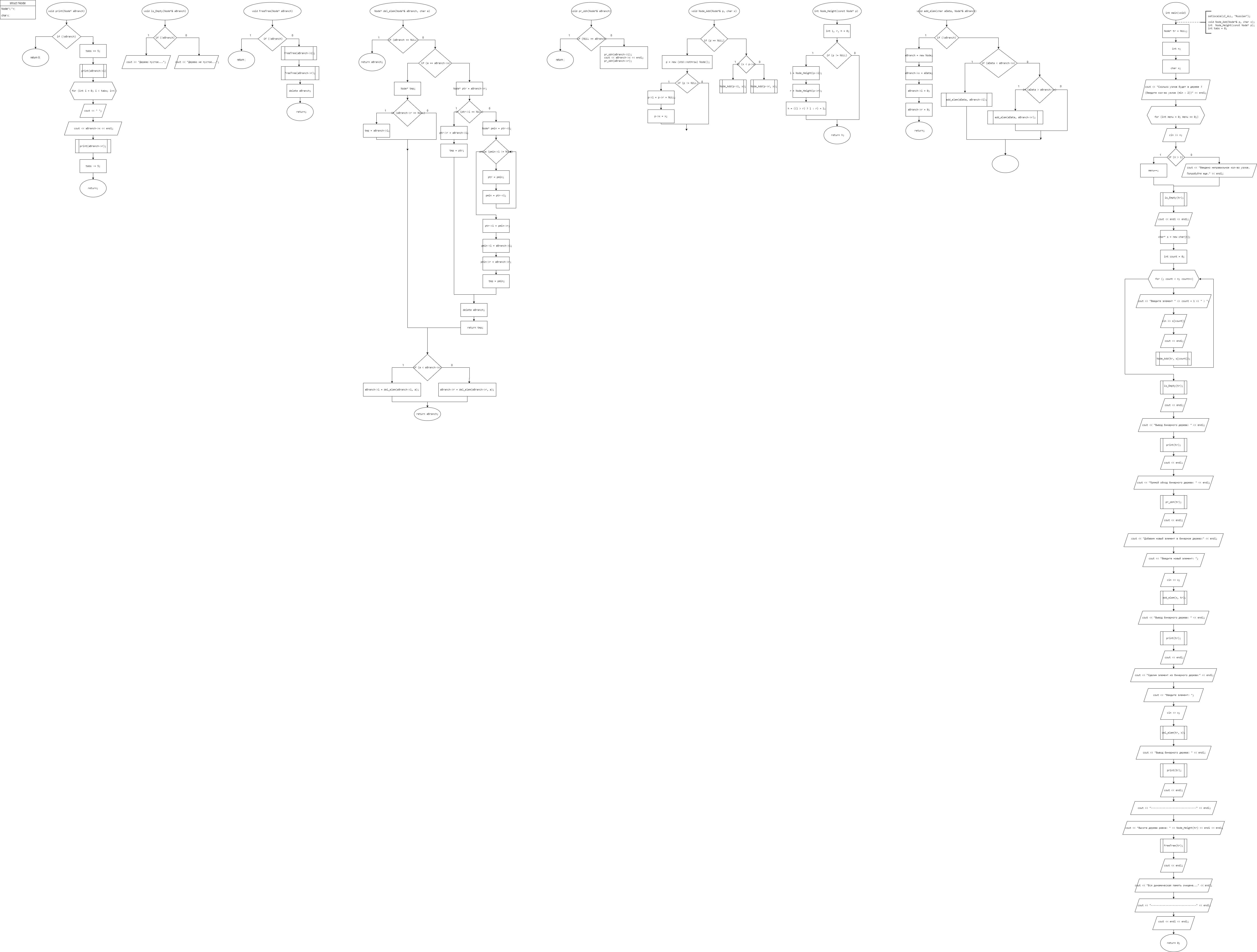


Рисунок 1