Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное учреждение высшего образования  
«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т  
по лабораторной работе**

Дисциплина: «Основы алгоритмизации и программирования»

Выполнил:  
студент группы ИВТ-22-2б  
Мифтахов Марат Ринатович

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

.

Пермь 2023

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.

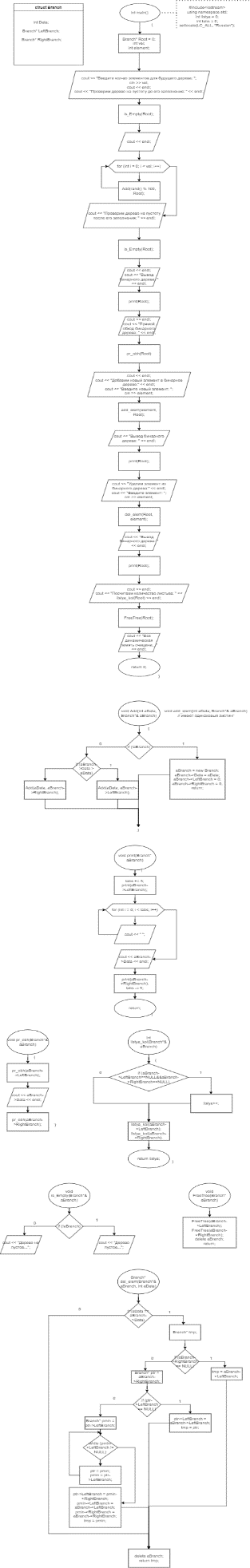
Анализ задачи:

1. int listya – количество листьев, int tabs – число пробелов для печати, int vel – количество элементов, int element – удаляемый/добавляемый элемент. Branch\*Root=0 – указатель типа структуры(обьект).

2. 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 ф-ия. Печать осуществляется также при помощи рекурсии с использованием добавления и удаления числа пробелов. Прямой обход осуществлен рекурсивно с левого поддерева – корня – правого поддерева. Ф-ия проверка на пустоту просто проверяет, есть ли хоть одна ветка. Возвращает сообщение пустое ли дерево. Также реализован метод очистки динамической памяти.

3. В ф-ии main() вызваны все методы и представлена их работа.

Блок-схема:



Код программы:

#include <iostream>

using namespace std;

int listya = 0;//счетчик листьев

int tabs = 0; //Для создания отступов

int kol\_vo = 0;

//Кол-во отступов высчитывается по кол-ву рекурсивного вхождения при выводе в фукцию print

//Структура ветки

struct Branch

{

int Data; //Поле данных

Branch\* LeftBranch; //УКАЗАТЕЛИ на соседние веточки

Branch\* RightBranch;

};

//Функция внесения данных

void Add(int aData, Branch\*& aBranch)

{

//Если ветки не существует

if (!aBranch)

{ //создадим ее и зададим в нее данные

aBranch = new Branch;

aBranch->Data = aData;

aBranch->LeftBranch = 0;

aBranch->RightBranch = 0;

return;

}

else //Иначе сверим вносимое

if (aBranch->Data > aData)

{ //Если оно меньше того, что в этой ветке - добавим влево

Add(aData, aBranch->LeftBranch);

}

else

{ //Иначе в ветку справа

Add(aData, aBranch->RightBranch);

};

}

//Функция вывода дерева

void print(Branch\* 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->LeftBranch); //Выведем ветку и ее подветки слева

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

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

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

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

return;

}

void pr\_obh(Branch\*& aBranch)

{

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

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

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

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

}

int listya\_kol(Branch\*& aBranch)

{

if (NULL == aBranch) return 0; //Если дерева нет, выходим

if (aBranch->LeftBranch==NULL&&aBranch->RightBranch==NULL)

{

listya++;

}

listya\_kol(aBranch->LeftBranch); //Обошли левое поддерево

listya\_kol(aBranch->RightBranch); //Обошли правое поддерево

return listya;

}

void add\_elem(int aData, Branch\*& aBranch)

{

if (!aBranch)

{

aBranch = new Branch;

aBranch->Data = aData;

aBranch->LeftBranch = 0;

aBranch->RightBranch = 0;

return;

}

else

{

if (aData < aBranch->Data) {

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

}

else if (aData > aBranch->Data) {

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

}

}

}

void is\_Empty(Branch\*& aBranch)

{

if (!aBranch)

{

cout << "Дерево пустое...";

}

else

{

cout << "Дерево не пустое...";

}

}

void FreeTree(Branch\* aBranch)

{

if (!aBranch) return;

FreeTree(aBranch->LeftBranch);

FreeTree(aBranch->RightBranch);

delete aBranch;

return;

}

Branch\* del\_elem(Branch\*& aBranch, int aData) {

if (aBranch == NULL)

return aBranch;

if (aData == aBranch->Data) {

Branch\* tmp;

if (aBranch->RightBranch == NULL)

tmp = aBranch->LeftBranch;

else {

Branch\* ptr = aBranch->RightBranch;

if (ptr->LeftBranch == NULL) {

ptr->LeftBranch = aBranch->LeftBranch;

tmp = ptr;

}

else {

Branch\* pmin = ptr->LeftBranch;

while (pmin->LeftBranch != NULL) {

ptr = pmin;

pmin = ptr->LeftBranch;

}

ptr->LeftBranch = pmin->RightBranch;

pmin->LeftBranch = aBranch->LeftBranch;

pmin->RightBranch = aBranch->RightBranch;

tmp = pmin;

}

}

delete aBranch;

return tmp;

}

else if (aData < aBranch->Data)

aBranch->LeftBranch = del\_elem(aBranch->LeftBranch, aData);

else

aBranch->RightBranch = del\_elem(aBranch->RightBranch, aData);

return aBranch;

}

int main()

{

setlocale(LC\_ALL, "rus");

Branch\* Root = 0;

int vel;

int element;

cout << "Введите кол-во элементов для будущего дерева: ";

cin >> vel;

cout << endl;

cout << "Проверим дерево на пустоту до его заполнения: " << endl;

is\_Empty(Root);

cout << endl;

for (int i = 0; i < vel; i++)

{

Add(rand() % 100, Root);

}

cout << "Проверим дерево на пустоту после его заполнения: " << endl;

is\_Empty(Root);

cout << endl;

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

print(Root);

cout << endl;

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

pr\_obh(Root);

cout << endl;

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

cout << "Введите новый элемент: ";

cin >> element;

add\_elem(element, Root);

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

print(Root);

cout << endl;

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

cout << "Введите элемент: ";

cin >> element;

del\_elem(Root, element);

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

print(Root);

cout << endl;

cout << "Посчитаем количество листьев: " << listya\_kol(Root) << endl;

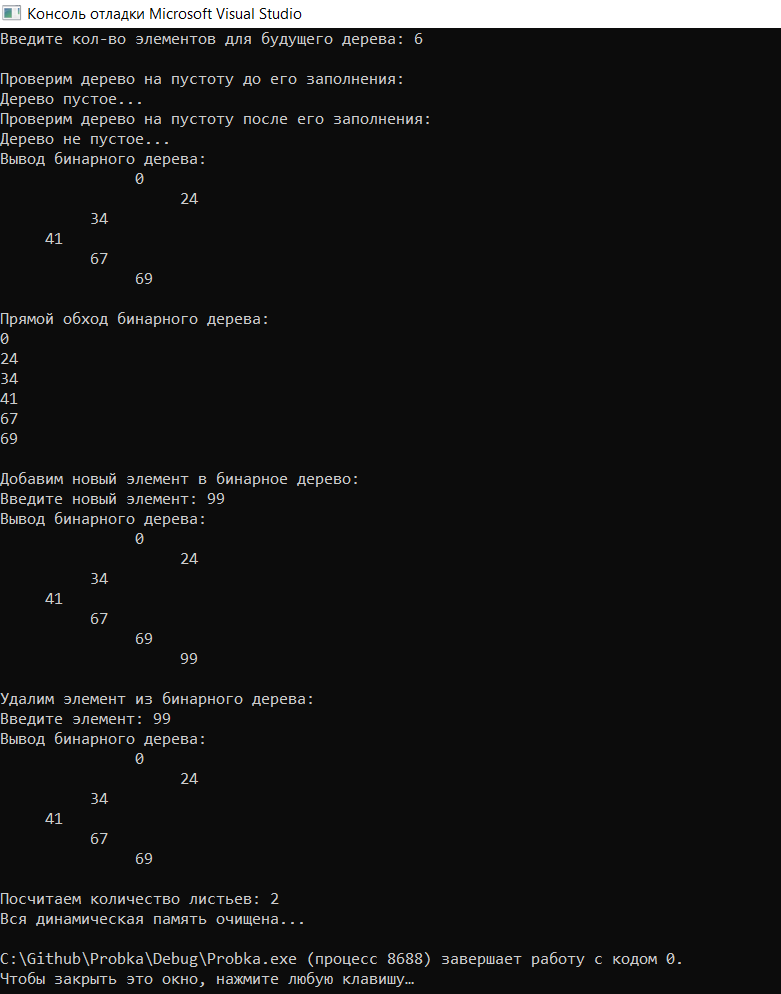
FreeTree(Root);

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

return 0;

}

Скриншоты результатов:



Анализ результатов:

Программа работает корректно. Выводится идеально сбалансированное дерево, все ф-ии работают как надо.