МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарные деревья

Студентка гр. 9304 _____ Селезнёва А.В.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с понятием бинарного дерева. Реализовать бинарное дерево для решения поставленной задачи на языке программирования C++.

Задание.

Вариант – 5у.

Заданы два бинарных дерева *b1* и *b2* типа *BT* с произвольным типом элементов. Проверить:

- подобны ли они (два бинарных дерева ***подобны***, если они оба пусты либо они оба непусты и их левые поддеревья подобны и правые поддеревья подобны);

- равны ли они (два бинарных дерева ***равны***, если они подобны и их соответствующие элементы равны);

- зеркально подобны ли они (два бинарных дерева ***зеркально подобны***, если они оба пусты либо они оба непусты и для каждого из них левое поддерево одного подобно правому поддереву другого);

- симметричны ли они (два бинарных дерева ***симметричны***, если они зеркально подобны и их соответствующие элементы равны).

Выполнение работы.

На вход программа получает две строки, каждая из которых содержит скобочную запись бинарного дерева.

Функция *bool IsCorrect()* проверяет корректность введенных данных. Если данные корректны, создается два бинарных дерева и вызываются четыре метода, которые определяют, подобны, равны, зеркально подобны или симметричны ли деревья. Результат работы методов выводятся в консоль.

Класс BinTreeNode:

Класс содержит публичные поля *left* и *right*, которые хранят умные указатели std::shared_ptr на левое и правое поддеревья соответственно, а также данные *data* шаблонного типа. Конструктор реализован по умолчанию.

Класс BinTree:

Класс содержит публичное поле head – умный указатель на корень дерева. В классе реализованы методы *CallAreTreesSimilar(), CallAreTreesEqual(), CallAreTreesMirrored(), CallAreTreesSymmetrical(),* которые вызывают методы *AreTreesSimilar(), AreTreesEqual(), AreTreesMirrored(), AreTreesSymmetrical()*. Они получают на вход указатели на корень каждого из деревьев и определяют, подобны, равны, зеркально подобны или симметричны ли деревья путем обхода каждого дерева. Также были реализованы конструкторы копирования и перемещения; перегружены операторы копирующего присваивания и перемещающего присваивания. Для этого был создан метод *copyTree()*. Конструктор создания дерева вызывает метод *CreateBinTreeNode()*, который создает дерево и возвращает умный указатель на корень дерева.

Разработанный программный код находится в приложении А.

На рисунке 1 изображено графическое представление двух бинарных деревьев (a(b)(c(e)) и (d(k(m))(f)) соответственно.

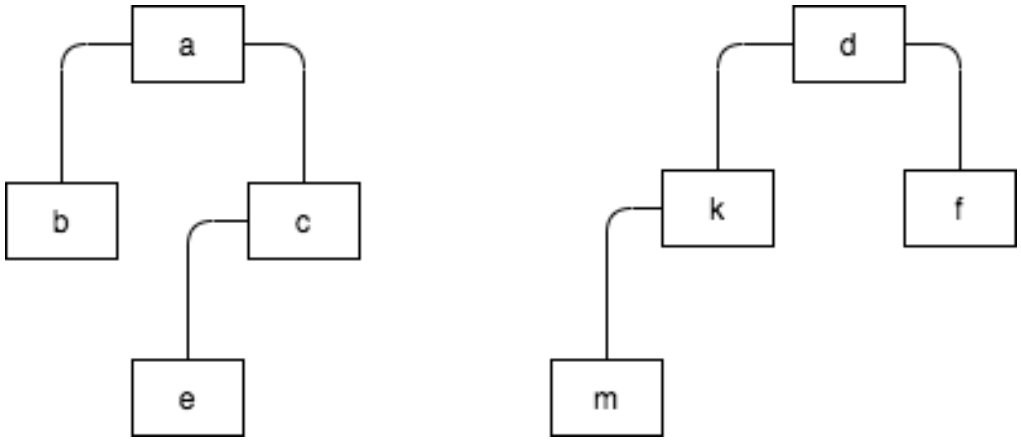

Рисунок 1 – Графическое представление двух бинарных деревьев

Тестирование.

Тестирование осуществляется с помощью bash-скрипта ./script. Скрипт запускает программу и в качестве входных аргументов подает строки, прописанные в текстовых файлах, расположенных в папке ./Tests.

Результаты тестирования представлены в приложении Б.

Выводы.

В ходе выполнения лабораторной работы было реализовано бинарное дерево на языке программирования C++.

Разработана программа, создающая два бинарных дерева и проверяющая, подобны, равны, зеркально подобны или симметричны ли они. Реализация бинарного дерева через указатели обусловлена небольшим потреблением памяти относительно реализации через массив.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <string>
#include <memory>

template <typename T>
class BinTreeNode {
public:
    BinTreeNode(){}
    BinTreeNode(std::shared_ptr<BinTreeNode<T>> left,
std::shared_ptr<BinTreeNode<T>> right, T data) :left(left),
right(right), data(data){}
    std::shared_ptr<BinTreeNode<T>> left;
    std::shared_ptr<BinTreeNode<T>> right;
    T data;
};

template <typename T>
class BinTree{
public:
~BinTree() = default;

BinTree(std::string str){
    head = CreateBinTreeNode(str);
}

std::shared_ptr<BinTreeNode<T>>
CreateBinTreeNode(std::string& str){
    if(str.size() == 3){
```

```
            std::shared_ptr<BinTreeNode<T>> Node =
std::make_shared<BinTreeNode<T>>();
            Node->left = nullptr;
            Node->right = nullptr;
            Node->data = str[1];
            return Node;
        } else if (str.size() > 3){
            size_t i = 2;
            int quantity = 0;
            do{
                if(str[i]=='('){
                    quantity++;
                }
                else if(str[i] == ')'){
                    quantity--;
                }
                i++;
            }while(quantity);
            std::string lefts = str.substr(2, i-2);
            std::string rights = str.substr(i, str.size()-1-i);
            std::shared_ptr<BinTreeNode<T>> Node =
std::make_shared<BinTreeNode<T>>();
            Node->left = CreateBinTreeNode(lefts);
            Node->right = CreateBinTreeNode(rights);
            Node->data = str[1];

            return Node;
        } else {
            return nullptr;
        }
    }

    void insert(std::shared_ptr <BinTreeNode<T>> root, T elem){
        if(!root){
            root = std::make_shared<BinTreeNode<T>>();
```

```
            root->left = nullptr;
            root->right = nullptr;
            root->data = elem;
        } else if(elem < root->data){
            insert(root->left, elem);
        } else {
            insert(root->right, elem);
        }
    }

    void LKP(std::shared_ptr <BinTreeNode<T>> root){
        if (root != nullptr){
            LKP(root->left);
            std::cout << root->data;
            LKP(root->right);
        }
    }

    void CallAreTreesSimilar(BinTree<T> root2){
        std::cout << "Trees are similar: " <<
AreTreesSimilar(this->head, root2.head) << '\n';
    }


    void CallAreTreesEqual(BinTree<T> root2){
        std::cout << "Trees are equal: " << AreTreesEqual(this-
>head, root2.head) << '\n';
    }

    void CallAreTreesMirrored(BinTree<T> root2){
        std::cout << "Mirror trees: " << AreTreesMirrored(this-
>head, root2.head) << '\n';
    }

    void CallAreTreesSymmetrical(BinTree<T> root2){
```

```
        std::cout << "Symmetric trees: " <<
AreTreesSymmetrical(this->head, root2.head) << '\n';
    }

    bool AreTreesSimilar(std::shared_ptr <BinTreeNode<T>> root1,
std::shared_ptr <BinTreeNode<T>> root2){
        if((root1 == nullptr) && (root2 == nullptr)){
            return true;
        }
        else if(AreTreesSimilar(root1->left, root2->left) == true
&& AreTreesSimilar(root1->right, root2->right) == true){
            return true;
        }
        else{
            return false;
        }
    }


    bool AreTreesEqual(std::shared_ptr <BinTreeNode<T>> root1,
std::shared_ptr <BinTreeNode<T>> root2){
        if(root1 == nullptr && root2 == nullptr){
            return true;
        }
        else if((AreTreesEqual(root1->left, root2->left) == true)
&& (root1->data == root2->data) && (AreTreesEqual(root1->right,
root2->right) == true)){
            return true;
        }
        else{
            return false;
        }
    }
```

```
    bool AreTreesMirrored(std::shared_ptr <BinTreeNode<T>> root1,
std::shared_ptr <BinTreeNode<T>> root2){
        if( root1->left == nullptr && root2->right != nullptr  ||
root1->right != nullptr && root2->left == nullptr || root1->left
!= nullptr && root2->right == nullptr  || root1->right == nullptr
&& root2->left != nullptr ){
            return  false;
        }
        if(root1->left == nullptr && root1->right==nullptr &&
root2->left == nullptr && root2->right == nullptr){
            return true;
        }
        if(root1->left == nullptr && root1->right!=nullptr &&
root2->left != nullptr && root2->right == nullptr){
            return AreTreesMirrored(root1->right, root2->left);
        }
         if(root1->left != nullptr && root1->right==nullptr &&
root2->left == nullptr && root2->right != nullptr){
            return AreTreesMirrored(root1->left, root2->right);
        }
        return (AreTreesMirrored(root1->left, root2->right) &&
AreTreesMirrored(root1->right, root2->left));
    }


    bool AreTreesSymmetrical(std::shared_ptr <BinTreeNode<T>>
root1, std::shared_ptr <BinTreeNode<T>> root2){
         if(root1->data != root2->data || root1->left == nullptr
&& root2->right != nullptr  || root1->right != nullptr && root2-
>left == nullptr || root1->left != nullptr && root2->right ==
nullptr  || root1->right == nullptr && root2->left != nullptr ){
            return  false;
        }
        if(root1->left == nullptr && root1->right==nullptr &&
root2->left == nullptr && root2->right == nullptr){
```

```
            return true;
        }
        if(root1->left == nullptr && root1->right!=nullptr &&
root2->left != nullptr && root2->right == nullptr){
            return AreTreesSymmetrical(root1->right, root2-
>left);
        }
         if(root1->left != nullptr && root1->right==nullptr &&
root2->left == nullptr && root2->right != nullptr){
            return AreTreesSymmetrical(root1->left, root2-
>right);
        }
        return (AreTreesSymmetrical(root1->left, root2->right) &&
AreTreesSymmetrical(root1->right, root2->left));

    }

        BinTree(BinTree<T>&& tree){
            std::swap(tree.head, head);
        }
        BinTree<T>& operator=(BinTree<T>&& tree){ // перемещение
            head = std::move(tree.head);
            return *this;
        }
        BinTree(BinTree<T>& tree){
            head = copyTree(tree.head);
        }
        BinTree<T>& operator=(BinTree<T>& tree){ // копирование
            tree = copyTree(tree.head);
            return *this;
        }
        std::shared_ptr<BinTreeNode<T>>
copyTree(std::shared_ptr<BinTreeNode<T>> tree){
             if (tree->left != nullptr && tree->right != nullptr)
{
```

```
                    std::shared_ptr<BinTreeNode<T>> node(new
BinTreeNode<T>(copyTree(tree->left), copyTree(tree->right), tree-
>data));
                    return node;
                }
                if (tree->left == nullptr && tree->right != nullptr)
{
                    std::shared_ptr<BinTreeNode<T>> node(new
BinTreeNode<T>(nullptr, copyTree(tree->right), tree->data));
                    return node;
                }
                if (tree->left != nullptr && tree->right == nullptr)
{
                    std::shared_ptr<BinTreeNode<T>> node(new
BinTreeNode<T>(copyTree(tree->left), nullptr, tree->data));
                    return node;
                }
                if (tree->left == nullptr && tree->right == nullptr)
{
                    std::shared_ptr<BinTreeNode<T>> node(new
BinTreeNode<T>(nullptr, nullptr, tree->data));
                    return node;
                }
            return nullptr;
        }

        std::shared_ptr<BinTreeNode<T>> head;
    };

    bool IsCorrect (std::string str){
        if(str[0] != '('){
            return false;
        }

        int k1 = 0;
```

```
    int k2 = 0;
    for (int i = 0; i < str.size(); ++i){
        if(str[i] == '('){
            k1++;
        }else if(str[i] == ')'){
            k2++;
        }
    }
    if(k1 == k2){
        return true;
    } else {
        return false;
    }
}

int main(){
    std::string str1;
    std::string str2;

    std::cin >> str1;
    std::cin >> str2;

    if((IsCorrect(str1) && IsCorrect(str2)) == true){
        BinTree<char> Tree1(str1);
        BinTree<char> Tree2(str2);
        Tree1.CallAreTreesSimilar(Tree2);
        Tree1.CallAreTreesEqual(Tree2);
        Tree1.CallAreTreesMirrored(Tree2);
        Tree1.CallAreTreesSymmetrical(Tree2);

    } else {
        std::cout << "Incorrect trees\n";
    }

    return 0;
```

```
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Результат проверки
1.	(a(b(c))(g(v))) (a(b(c))(g(v)))	Trees are similar: 1 Trees are equal: 1 Mirror trees: 0 Symmetric trees: 0	correct
2.	(a(b)(c)) (a(c)(b))	Trees are similar: 1 Trees are equal: 0 Mirror trees: 1 Symmetric trees: 1	correct
3.	(a) (b)	Trees are similar: 1 Trees are equal: 0 Mirror trees: 1 Symmetric trees: 0	correct
4.	a c	Incorrect trees	correct
5.	((b) (c)	Incorrect trees	correct
6.	(a(c)) (a(b))	Trees are similar: 1 Trees are equal: 0 Mirror trees: 0 Symmetric trees: 0	correct
7.	(a) (a)	Trees are similar: 1 Trees are equal: 1 Mirror trees: 1 Symmetric trees: 1	correct