

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по практической работе №4**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Деревья**

Студент гр. 8304

Преподаватель

Ястребов И.М.

Фирсов М.А.

Санкт-Петербург

2019

## Цель работы

Научиться реализовать рандомизированное бинарное дерево поиска (РБДП) и основные функции работы с ним.

## Задание

Рассматриваются бинарные деревья с элементами типа Elem (в качестве Elem использовать char). Заданы перечисления узлов некоторого дерева b в порядке КЛП и ЛКП. Требуется:

- восстановить дерево b и вывести его изображение;
- перечислить узлы дерева b в порядке ЛПК.

## Описание алгоритма

1. Открывается файл с тремя входными строками.
2. Первая определяет вид представления графа во входной строке.
3. Вторая — дерево в строчной записи
4. Третья — вид представления графа в выходной строке.
5. Узлы графа создаются соответственно способу представления графа в строке
6. При выводе используется соответствующий способ обхода — ЛПК, КЛП или ЛКП.

Тестирование программы приведено в Приложении А, исходный код программы представлен в Приложении Б.

## Описание основных функций

1. `sIter` bracket\_closer(`sIter` begin)

Функция находит итератор соответствующей парной скобки для данной

- 2.

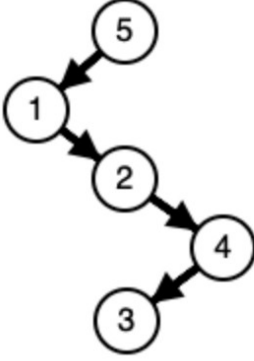
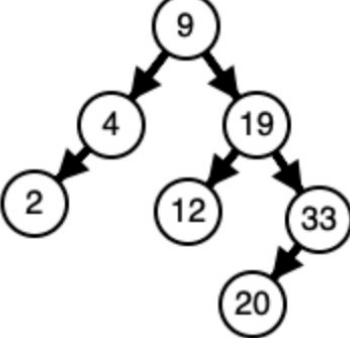
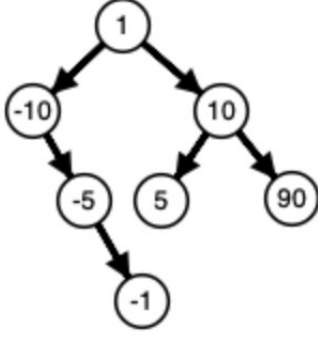
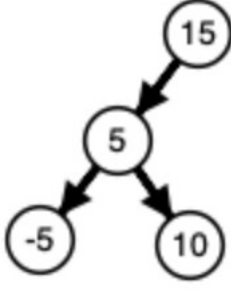
```
nodePtr<Elem> readTreeFromStringNLR(std::string&, sIter, sIter);  
nodePtr<Elem> readTreeFromStringLNR(std::string&, sIter, sIter);  
nodePtr<Elem> readTreeFromStringLRN(std::string&, sIter, sIter);  
void printTreeNLR(nodePtr<Elem> root);  
void printTreeLNR(nodePtr<Elem> root);  
void printTreeLRN(nodePtr<Elem> root);
```

Функции производят считывание и вывод графа в соответствующем виде.

**Вывод.**

Были получены навыки работы с деревьями, структура данных была реализована на языке программирования C++.

# **ПРИЛОЖЕНИЕ А** **Тестирование программы**

Входные данные	Выходные данные	Визуализация РБДП
1 2 3 4 5	(5) - - - (4) - - - - (3) - - (2) - (1)	
19 2 20 4 33 9 12	- - (33) - - - (20) - (19) - - (12) (9) - (4) - - (2)	
-10 -5 90 -1 1 5 10	- - (90) - (10) - - (5) (1) - - - (-1) - - (-5) - (-10)	
15 5 10 -5 10 5 15	Element 10 already exists Element 5 already exists Element 15 already exists (15) - - (10) - (5) - - (-5)	

## ПРИЛОЖЕНИЕ Б

### Файл main.cpp

```
#include <iostream>
#include <fstream>
#include <string>

#include "RBST.h"

int readFromFile(const std::string& str)
{
    std::string str1;
    int element;
    std::cout << "For file: " << str << std::endl;
    std::ifstream inputFile(str);
    if (!inputFile.is_open())
    {
        std::cout << "ERROR: file isn't open" << std::endl;
        return 0;
    }
    if (inputFile.eof())
    {
        std::cout << "ERROR: file is empty" << std::endl;
        return 0;
    }
    RandomBinarySearchTree<int> Tree;
    while (inputFile >> element)
        Tree.searchAndInsertElement(element);
    std::cout << "Do you want to add some elements? (y/n)" << std::endl;
    while (getchar() != 'n')
    {
        std::cout << "Input element:" << std::endl;
        std::cin >> element;
        Tree.searchAndInsertElement(element);
        getchar();
        std::cout << "Do you want to add some elements? (y/n)" <<
std::endl;
    }
    Tree.printTree(0);
    inputFile.close();
    return 0;
}

int main(int argc, char* argv[])
{
    if (argc == 1)
    {
        std::string str;
        std::cout << "Input file path:" << std::endl;
        std::getline(std::cin, str);
        readFromFile(str);
    }
    else readFromFile(argv[1]);
    return 0;
}
```

## Файл RBST.h

```
#pragma once
#include <random>

template <class Elem>
class RandomBinarySearchTree
{
private:
    struct Node
    {
        Elem element;
        std::shared_ptr<Node> left;
        std::shared_ptr<Node> right;
        int N;
        explicit Node (Elem value)
        {
            element = value,
            left = right = 0;
            N = 1;
        }
    };
    typedef std::shared_ptr<Node> link;
    link head;

    link search(link root, Elem value)
    {
        if (!root) return 0;
        if (value == root->element) return root;
        if (value < root->element) return search(root->left, value);
        else return search(root->right, value);
    }

    int getSize(link root)
    {
        if (!root) return 0;
        return root->N;
    }

    void fixN(link root)
    {
        root->N = getSize(root->left) + getSize(root->right) + 1;
    }

    link rotateRight(link root)
    {
        link tmp = root->left;
        if (!tmp) return root;
        root->left = tmp->right;
        tmp->right = root;
        tmp->N = root->N;
        fixN(root);
        return tmp;
    }

    link rotateLeft(link root)
    {
        link tmp = root->right;
        if (!tmp) return root;
        root->right = tmp->left;
        tmp->left = root;
        tmp->N = root->N;
        fixN(root);
        return tmp;
    }
};
```

```

    }

    link insertRoot(link root, Elem value)
    {
        if (!root) return std::unique_ptr<Node>(new Node(value));
        if (root->element > value)
        {
            root->left = insertRoot(root->left, value);
            return rotateRight(root);
        }
        else
        {
            root->right = insertRoot(root->right, value);
            return rotateLeft(root);
        }
    }

    link insert(link root, Elem value)
    {
        if (!root) return std::unique_ptr<Node>(new Node(value));
        if (rand()%(root->N+1) == 0) return insertRoot(root, value);
        if (root->element > value) root->left = insert(root->left, value);
        else root->right = insert(root->right, value);
        fixN(root);
        return root;
    }

    void print(link root, int i){
        if (root->right != 0) print(root->right, i+1);
        for (int j = 0 ; j < i; ++j)
            std::cout << " - ";
        std::cout << "(" << root->element << ")" << std::endl;
        if (root->left != 0) print(root->left, i+1);
    }

public:
    RandomBinarySearchTree()
    {
        head = 0;
    }

    void searchAndInsertElement(Elem value)
    {
        if (!search(head, value)) head = insert(head, value);
        else std::cout << "Element " << value << " already exists" <<
std::endl;
    }

    void printTree(int i)
    {
        print(head, i);
    }
};

```