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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: БДП.

Студент гр. 9384 Мосин К.К.

Преподаватель Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

Реализовать структуру бинарного дерева поиска, далее БДП, заполнить ее и записать в файл.

Задание.

ВАРИАНТ 11.

БДП: случайное БДП с рандомизацией.

По заданной последовательности элементов Elem построить структуру данных определенного типа - БДП или хеш-таблицу;

Записать в файл элементы построенного БДП в порядке их возрастания; вывести построенное БДП на экран.

Выполнение работы.

Для реализации структуры БДП создаются два класса: Tree для работы с деревом и Node для работы с узлом. Для того, чтобы не устанавливать геттеры и сеттеры в классе Node, класс Tree объявляется классом-другом; такие методы, как print_tree и write_tree помечены protected доступом, а также поля data, parent, left и right. Реализацию рандомизации можно опустить, так как реализован ввод с клавиатуры или файла, где пользователь может сгенерировать данные сам. Заполнение структуры осуществляется посредством бинарного отношения, и в данном случае “<” и “>”. Если дерево пустое, вводимый элемент становится корнем, далее, если очередной элемент меньше, то он становится ребенком слева для данного узла. Аналогично, если элемент больше, то он отправляется на правую сторону.

Тестирование.

Результаты тестирования представлены в табл. 2.

Таблица 2 – Результаты тестирования

Входные данные	Выходные данные
console 3 3 2 1	1 2 3 outfile: 1 2 3
console 5 1 2 3 4 5	1 2 3 4 5 outfile: 1 2 3 4 5
console 5 3 3 3 3 2	2 3 3 3 3 outfile: 2 3 3 3 3

Выводы.

В ходе выполнения лабораторный работы была реализовано БДП.

ПРИЛОЖЕНИЕ
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include "tree.h"


int main(){
    std::string temp;
    std::cout << "Type console to write tree in console or type path to read file\n";
    std::cin >> temp;
    if(temp == "console"){
        Tree<int> tree;
        int count;
        std::cout << "Type count: ";
        std::cin >> count;

        std::cout << "Type data: ";
        for(int i = 0; i < count; i++){
            std::cin >> temp;
            tree.insert_node(std::stoi(temp));
        }
        tree.render_node();
        tree.write_node("output.txt");
    }

    else{
        std::ifstream file(temp);
        if(file.is_open()){
            while(getline(file, temp)){
                Tree<int> tree;
                std::istringstream str(temp);
                std::string data;
                while(getline(str, data, ' ')){
                    tree.insert_node(std::stoi(data));
                }
                tree.render_node();
                tree.write_node("output.txt");
            }
        }
        else{
            throw std::runtime_error("type valid path\n");
        }
    }


    return 0;
```

```
}
```

Название файла: node.h

```
#ifndef NODE_H
#define NODE_H


template <typename T>
class TreeNode{
   friend class Tree<T>;
public:
   TreeNode(T);
   ~TreeNode();
   T get_data();


protected:
   void print_tree(int = 0);
   void write_tree(std::ofstream&);


protected:
   T data;
   TreeNode *parent;
   TreeNode *left;
   TreeNode *right;
};


template <typename T>
TreeNode<T>::TreeNode(T data) : data(data), parent(nullptr), left(nullptr), right(nullptr) {}


template <typename T>
T TreeNode<T>::get_data(){
   return data;
}


template <typename T>
void TreeNode<T>::print_tree(int level){
   if(this->left){
      this->left->print_tree(level+1);
   }


   for(int i = 0; i < level; i++){
      std::cout << '\t';
   }
   std::cout << this->data << std::endl;

   if(this->right){
```

```
        this->right->print_tree(level+1);
    }
}


template <typename T>
void TreeNode<T>::write_tree(std::ofstream &file){
    if(this->left){
        this->left->write_tree(file);
    }
    file << this->data << " ";
    if(this->right){
        this->right->write_tree(file);
    }
}


#endif
```

Название файла: tree.h

```
#ifndef TREE_H
#define TREE_H


template <typename T> class Tree;
#include "node.h"


template <typename T>
class Tree{
public:
    Tree();
    void insert_node(T);
    void render_node();
    void write_node(std::string path);


private:
    TreeNode<T> *root;
};


template <typename T>
Tree<T>::Tree() : root(nullptr) {}


template <typename T>
void Tree<T>::insert_node(T data){
    TreeNode<T> *node = new TreeNode<T>(data);
    TreeNode<T> *ptr = this->root;
    TreeNode<T> *temp = nullptr;
```

```
    while(ptr){
        temp = ptr;
        if(data < ptr->get_data()){
            ptr = ptr->left;
        }
        else{
            ptr = ptr->right;
        }
    }
    node->parent = temp;

    if(!temp){
        this->root = node;
    }
    else if(data < temp->get_data()){
        temp->left = node;
    }
    else{
        temp->right = node;
    }
}


template <typename T>
void Tree<T>::render_node(){
    if(this->root){
        this->root->print_tree();
    }
    else{
        std::cout << "NULL" << std::endl;
    }
}


template <typename T>
void Tree<T>::write_node(std::string path){
    if(this->root){
        std::ofstream file(path, std::ios::app);
        if(file.is_open()){
            this->root->write_tree(file);
            file << std::endl;
        }
        else{
            std::cout << "file " << path << " could not open" << std::endl;
        }
    }
}


#endif
```