

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

ОТЧЕТ
по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: АВЛ-дерево

Студентка гр. 9304

Селезнёва А.В.

Преподаватель

Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

Изучить структуру данных АВЛ-дерево. Реализовать АВЛ-дерево на языке программирования C++.

Задание.

Вариант – 17.

БДП: АВЛ-дерево.

По заданной последовательности элементов Elem построить структуру данных определённого типа – БДП.

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

Выполнение работы.

На вход программа получает строку, корректность которой проверяет функции *bool IsCorrect()*. Если элемент не является целым числом, то он не будет записан в вектор. Если полученный вектор является пустым, программа завершит свою работу, иначе он (вектор) будет отсортирован с помощью *std::sort()* из библиотеки *algorithm* для удобства создания дерева. После создания дерева будут вызваны два метода *print_avl()* и *writing()*. Первый из них выводит построенное дерево в консоль в наглядном виде, а второй записывает в файл элементы дерева в порядке их возрастания.

Класс Node:

Класс содержит публичные поля *left* и *right*, которые хранят умные указатели *std::shared_ptr* на левое и правое поддеревья соответственно, а также данные *key* типа *int*. Данные *height* типа *int* содержат уровень, на котором находится node.

Класс AVL_Tree:

Класс содержит приватное поле *Head* – умный указатель на корень дерева, а также приватные методы *LKP()*, *print_node()*, *create_AVL()*, которые вызываются публичными методами *writing()*, *print_AVL()* и конструктором *AVL_Tree()* соответственно. Метод *LKP()* осуществляет инфиксный обход по дереву и добавляет каждый элемент дерева в вектор. В методе *writing()* полученный на предыдущем шаге вектор записывается в файл в порядке возрастания. Метод *print_node()* обходит дерево и выводит его на экран. Метод *create_AVL()* создает ноды дерева и возвращает указатель на вершину, который записывается в поле *Head* в конструкторе класса *AVL_Tree()*.

Разработанный программный код находится в приложении А.

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 была изучена и реализована структура данных АВЛ-дерево на языке программирования C++ с использованием умных указателей.

Была разработана программа, которая по заданной последовательности элементов создает и наглядно выводит АВЛ-дерево на экран, а также записывает его элементы в порядке возрастания в файл.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Lb5.cpp

```
#include <iostream>
#include <memory>
#include <vector>
#include <algorithm>
#include <iomanip>
#include <fstream>
#include <string>

class Node {
public:
    Node(int k) {
        height = k;
    }
    std::shared_ptr<Node> left;
    std::shared_ptr<Node> right;
    int height;
    int key = 0;
};

class AVL_Tree {
public:
    AVL_Tree(std::vector<int> v) {
        Head = this->create_AVL(v, 0);
    }
    void print_avl() {
        print_node(this->Head);
    }
    void writing() {
        std::vector<int> vec;
        vec = LKP(this->Head, vec);

        std::ofstream avl("AVL.txt");
        for (auto i : vec) {
            avl << i << ' ';
        }
        avl.close();
    }

private:
    std::shared_ptr<Node> Head;
    std::vector<int> LKP(std::shared_ptr<Node> ptr,
std::vector<int> vec) {
        if (ptr != nullptr) {
```

```

        vec = LKP(ptr->left, vec);
        vec.push_back(ptr->key);
        vec = LKP(ptr->right, vec);
    }
    return vec;
}

void print_node(std::shared_ptr<Node> ptr, int i = 0) {
    if (ptr != nullptr) {
        if (ptr->right) {
            print_node(ptr->right, i + 4);
        }
        if (i) {
            std::cout << std::setw(i) << ' ';
        }
        if (ptr->right) {
            std::cout << " /\n" << std::setw(i) << '
';

        }
        std::cout << ptr->key << "\n ";
        if (ptr->left) {
            std::cout << std::setw(i) << ' ' << "
\\n";

            print_node(ptr->left, i + 4);
        }
    }
}

std::shared_ptr<Node> create_AVL(std::vector<int> v, int
h) {
    if (v.size() == 1) {
        std::shared_ptr<Node> node =
std::make_shared<Node>(h);
        node->left = nullptr;
        node->right = nullptr;
        node->key = v[0];
        return node;
    }
    else if (v.empty()) {
        return nullptr;
    }
    else {
        size_t i = v.size() / 2;
        std::vector<int> v_left;
        std::vector<int> v_right;
        for (size_t z = 0; z < i; ++z) {
            v_left.push_back(v.at(z));
        }
        for (size_t z = i+1; z < v.size(); ++z) {
            v_right.push_back(v.at(z));
        }
        std::shared_ptr<Node> node =
std::make_shared<Node>(h);
        node->left = create_AVL(v_left, h+1);
        node->right = create_AVL(v_right, h+1);
    }
}

```

```

        node->key = v[i];
        return node;
    }
}

};

typedef int elem;

bool isCorrect(const std::string str) {
    size_t i = 0;
    if (str.at(0) == '-' && i != str.size() - 1) {
        ++i;
    }
    while (i < str.size()) {
        if (isdigit(str.at(i))) {
            ++i;
        }
        else {
            return false;
        }
    }
    return true;
}

int main() {
    std::vector<elem> out_1;
    std::string x;
    int i = 0;
    do {
        std::cin >> x;
        if (isCorrect(x)) {
            if (x.at(0) == '-') {
                std::string x_1 = x.substr(1, x.size() -
1);
                out_1.push_back(-atoi(x_1.c_str()));
            }
            else {
                out_1.push_back(atoi(x.c_str()));
            }
            i++;
        }
    } while (getchar() != '\n');

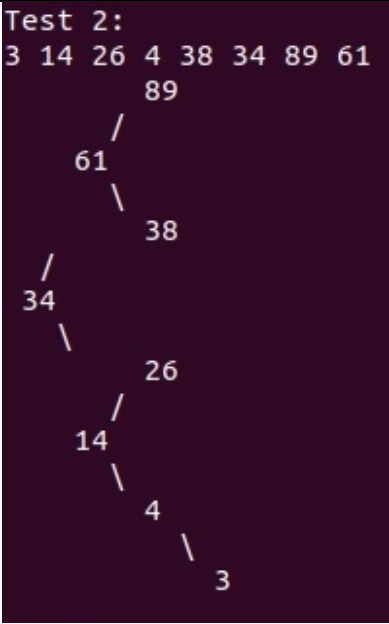
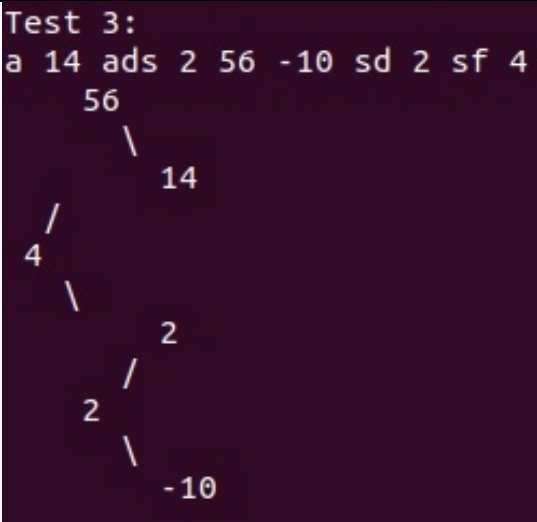
    if (!out_1.empty()) {
        std::sort(out_1.begin(), out_1.end());
        AVL_Tree tree(out_1);
        tree.print_avl();
        tree.writing();
    }
    return 0;
}

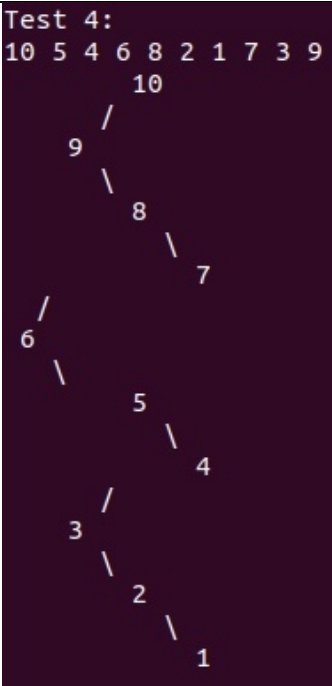
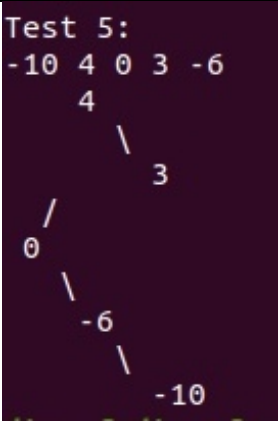
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Результат проверки
1.	a		correct
2.	3 14 26 4 38 34 89 61		correct
3.	a 14 ads 2 56 -10 sd 2 sf 4		correct

4.	10 5 4 6 8 2 1 7 3 9		correct
5.	-10 4 0 3 -6		correct