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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Программирование алгоритмов с бинарными деревьями

Студент гр. 9381 _____ Чухарев И.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы

Познакомиться с иерархической структурой данных в виде бинарного дерева, написать реализацию на языке программирования C++.

Постановка задачи

Вариант 8-д.

(Обратная задача.) Для заданного бинарного дерева с произвольным типом элементов:

- получить лес, естественно представленный этим бинарным деревом;
- вывести изображение бинарного дерева и леса;
- перечислить элементы леса в горизонтальном порядке (в ширину).

Основные теоретические положения

Бинарное дерево — иерархическая структура данных, в которой каждый узел имеет не более двух потомков. Как правило, первый называется родительским узлом, а дети называются левым и правым наследниками. Бинарное дерево не является упорядоченным ориентированным деревом

Описание алгоритма

Вывод изображения бинарного дерева реализован в функции treeToString(). Производим обход бинарного дерева, записывая в строку знаки табуляции и информации в звене, пока не дойдём до конца ветки. Затем производим вывод текущей инфомации и возврат по уровням рекурсии.

Обход в ширину реализован на базе стека, проходим звенья бинарного дерева слева направо, сверху вниз, добавляя и удаляя со стека вершину, пока не дойдём до конца. В процессе выводим информацию.

Описание реализованных классов:

BinaryNode — звено бинарного дерева. Приватные поля pBinaryNode left_ и pBinaryNode right_ хранят указатели на детей текущего звена, char data_ хранит символ звена.

BinaryTree — бинарное дерево. Приватные поля: pBinaryNode root_ — указатель на корень дерева, std::string stringTree_ - строковое представление дерева.

Описание реализованных функций:

- *int getAction()* - возвращает int — выбор пользователя, считанный с клавиатуры.
- void outputHelp() - вывод подсказки для пользователя.
- void bfSearch() - производит обход в ширину дерева и выводит звенья.
- void execConvert(pBinaryTree tree) - принимает указатель на дерево, которое необходимо представить в ввиде леса и производит обработку согласно заданию.
- *void treeToString(pBinaryNode node, std::string string)* — принимает pBinaryNode node — указатель на текущее звено, std::string &input — ссылка на строку, необходимую для записи текущих данных. Производит обход дерева и вывод представления, повернутого на 90 градусов.

Описание методов класса BinaryTree:

- *void initTree(std::string input)* — принимает строку с записью исходного дерева. Запускает функцию для рекурсивного обхода input readTree.
- *pBinaryNode getRoot()* - возвращает pBinaryNode - указатель на корень дерева.
- *void deleteTree(pBinaryNode node)* — принимает pBinaryNode node — указатель на корень дерева, которое требуется удалить.

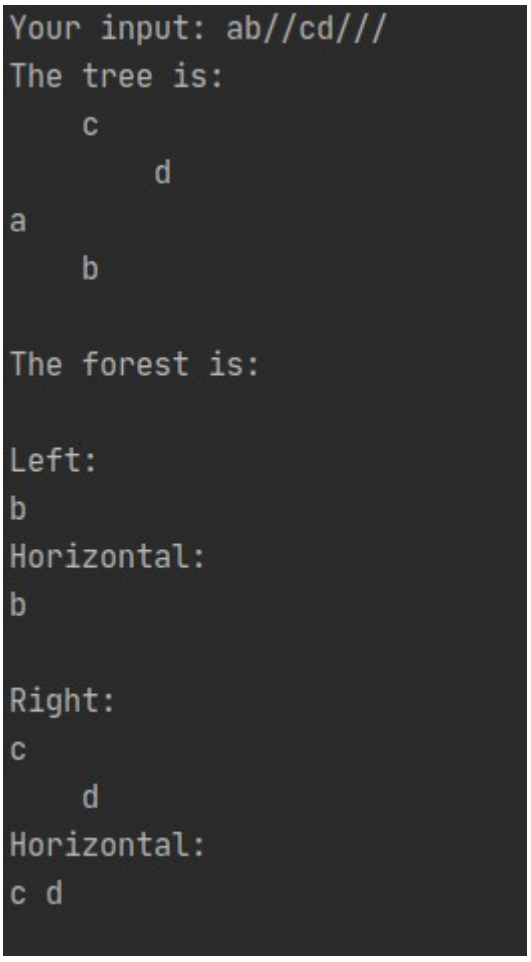
Производит его рекурсивный обход и очищает память, выделенную для каждого звена.

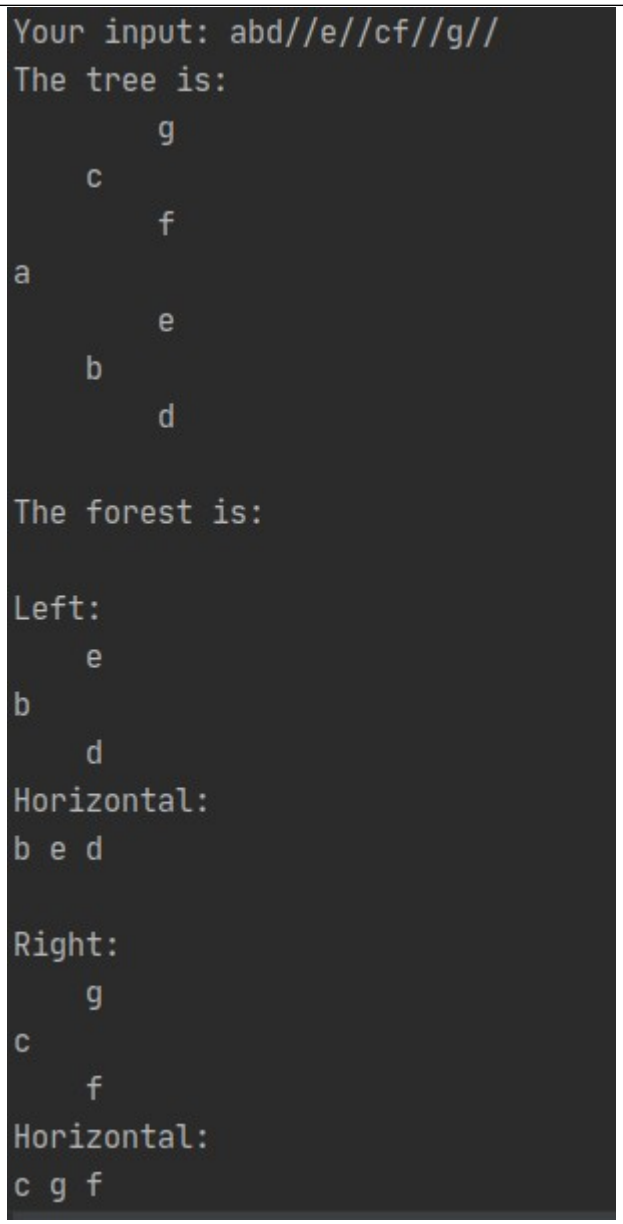
- *void toString(pBinaryNode node, std::string &input, int level)* — принимает pBinaryNode node — указатель на текущий элемент, std::string &input — ссылка на строку, в которую записывается результат, int level — глубина звена. Производит рекурсивный обход дерева и запись данных.
- *pBinaryNode readTree(std::string &input)* — принимает std::string &input — исходную строку с записью дерева, возвращает pBinaryNode - звено дерева. Производит рекурсивный обход строки и формирует дерево.
- friend std::ostream& operator<<(std::ostream &stream, const BinaryTree &tree) — переопределённый оператор потока вывода. std::ostream &stream — поток вывода, const BinaryTree &tree — ссылка на дерево, которое необходимо вывести. Выводит строковое представление stringTree_.

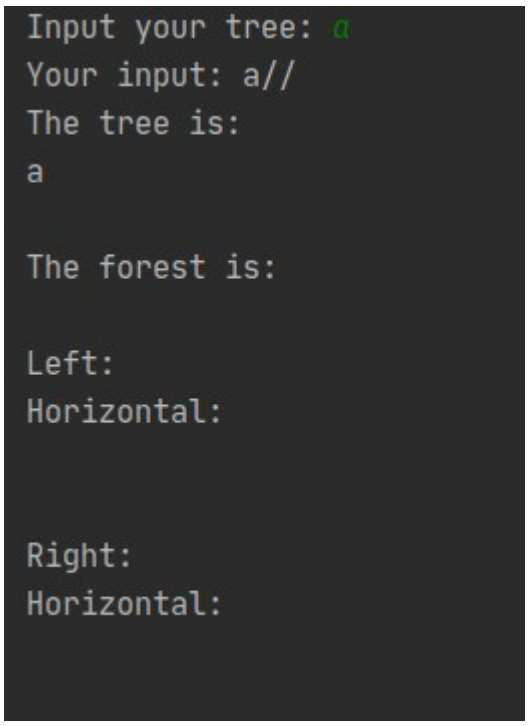
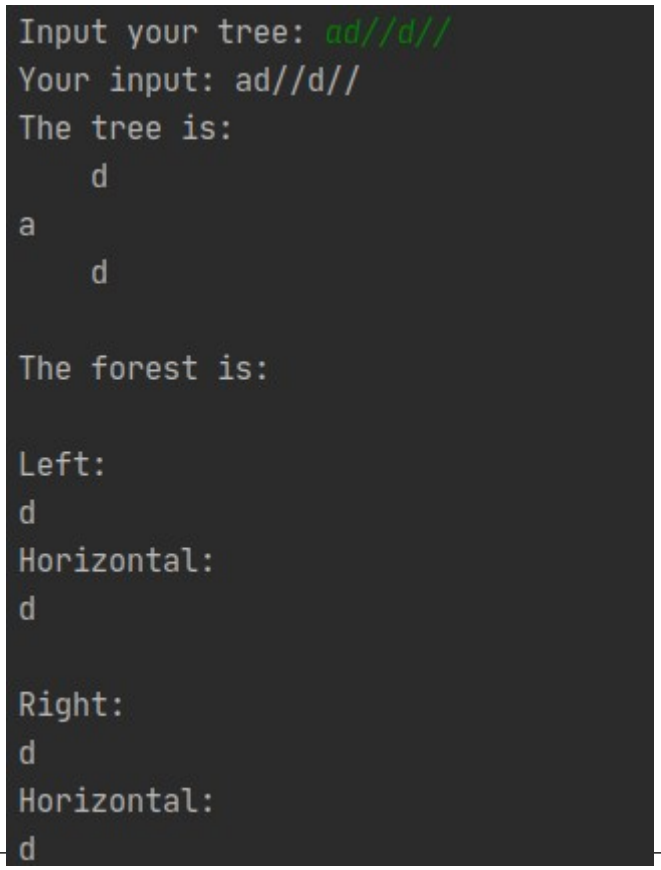
Описание методов класса BinaryNode:

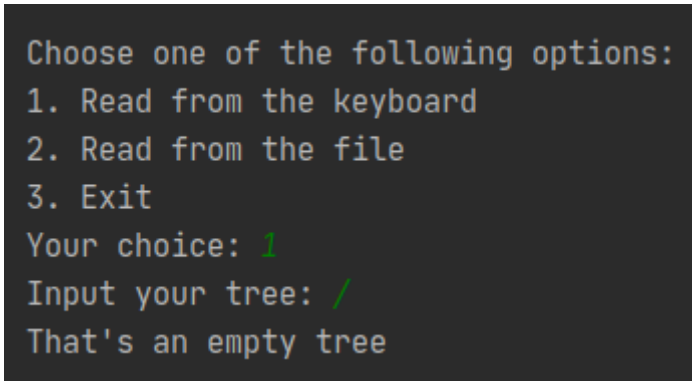
- *pBinaryNode getLeft()* - возвращает *pBinaryNode* - указатель на левое плечо текущего звена.
- *void setLeft(pBinaryNode left)* — принимает *pBinaryNode* - указатель на звено, которое требуется установить в качестве леовго плеча.
- *void setRight(pBinaryNode right)* — принимает *pBinaryNode* — указатель на звено, которое требуется установить в качестве правого плеча.
- *pBinaryNode getRight()* - возвращает *pBinaryNode* - указатель на правое плечо текущего звена.
- *char getData()* - возвращает char — символ текущего звена.

Тестирование

Входные данные	Исходные данные
1. ab//cd///	

2. abd//e//cf//g//	 Your input: abd//e//cf//g// The tree is: g c f a e b d The forest is: Left: e b d Horizontal: b e d Right: g c f Horizontal: c g f

3. a	 Input your tree: a Your input: a// The tree is: a The forest is: Left: Horizontal: Right: Horizontal:
4. ad//d//	 Input your tree: ad//d// Your input: ad//d// The tree is: d a d The forest is: Left: d Horizontal: d Right: d Horizontal: d

5. /	

Выводы

В ходе работы были приобретены навыки работы с бинарными деревьями, написана реализация на языке прогаммирования C++.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb3.cpp

```
#include <iostream>
#include <fstream>
#include <stack>

class BinaryNode;
class BinaryTree;

using pBinaryNode = BinaryNode*;
using pBinaryTree = BinaryTree*;

// Node of the binary tree.
class BinaryNode {
    pBinaryNode left_;
    pBinaryNode right_;
    char data_;

public:
    BinaryNode() : left_(nullptr), right_(nullptr), data_('\0') {}

    explicit BinaryNode(char symbol) : BinaryNode() {
        data_ = symbol;
    };

    // Setters
    void setLeft(pBinaryNode left) {
        left_ = left;
    }

    void setRight(pBinaryNode right) {
        right_ = right;
    }

    // Getters
```

```
        pBinaryNode getRight() {
            return right_;
        }

        char getData() {
            return data_;
        }

        pBinaryNode getLeft() {
            return left_;
        }
    };

    // Binary tree, node is a BinaryNode
    class BinaryTree {
        pBinaryNode root_;
        std::string stringTree_; // string representation of the tree

    public:
        BinaryTree() : root_(nullptr) {};

        ~BinaryTree() {
            deleteTree(root_);
        }

        // Starts recursion read of the tree
        void initTree(std::string &input) {
            if (input.empty() || input == "\n")
                return;

            root_ = new BinaryNode();
            root_ = readTree(input);
            toString(getRoot(), stringTree_, 1); // get the string
representation
        }

        friend std::ostream& operator<<(std::ostream &stream, const
BinaryTree &tree);
```

```
        // Getters
        pBinaryNode getRoot() {
            return root_;
        }

    private:
        // Get the string representation of the tree
        void toString(pBinaryNode node, std::string &string, int level)
{
            if (!node) {
                string += '/';
                return;
            }

            string += node->getData();
            toString(node->getLeft(), string, level + 1);
            toString(node->getRight(), string, level + 1);
        }

        // Recursion tree memory free
        void deleteTree(pBinaryNode node) {
            if (!node)
                return;

            deleteTree(node->getLeft()); // Go to left
            deleteTree(node->getRight()); // Go to right
            delete node;
        }

        // Recursion read the tree
        pBinaryNode readTree(std::string &input) {
            if (input.empty())
                return nullptr;

            char symbol = input[0];
            input = input.substr(1);

            if (symbol == '/') {
                return nullptr;
```

```
            } else {
                auto buf = new BinaryNode(symbol);
                buf->setLeft(readTree(input));
                buf->setRight(readTree(input));
                return buf;
            }
        }
    };

    void treeToString(pBinaryNode node, std::string string) {
        if (!node)
            return;

        if (!node->getLeft() && !node->getRight()) { // the end of the
branch
            std::cout << string << node->getData() << '\n';
            return;
        }

        std::string temp = string;
        if (node->getRight()) { // go to the right
            temp += '\t';
            treeToString(node->getRight(), temp);
        }

        std::cout << string << node->getData() << '\n';
        if (node->getLeft()) { // go to the left
            string += '\t';
            treeToString(node->getLeft(), string);
        }
    }

    std::ostream& operator<<(std::ostream &stream, const BinaryTree
&tree) {
        return stream << tree.stringTree_;
    }

    void outputHelp() {
        std::cout << "Choose one of the following options: " << '\n' <<
```

```
            "1. Read from the keyboard" << '\n' <<
            "2. Read from the file" << '\n' <<
            "3. Exit" << '\n' <<
            "Your choice: ";
}

int getAction() {
    int action;
    outputHelp();
    std::cin >> action;
    return action;
}

void bfSearch(pBinaryNode node) { // breadth-first traversal
    if (!node)
        return;

    std::stack <pBinaryNode> nodes;
    nodes.push(node);

    while (!nodes.empty()) {
        auto temp = nodes.top();
        nodes.pop();

        if (temp) {
            std::cout << temp->getData() << ' ';
            nodes.push(temp->getLeft());
            nodes.push(temp->getRight());
        }
    }
}

void execConvert(pBinaryTree tree) {
    std::cout << "Your input: ";
    std::cout << *tree << '\n';

    std::cout << "The tree is: " << '\n';
    treeToString(tree->getRoot(), "");
```

```
    std::cout << '\n';
    std::cout << "The forest is: " << '\n' << '\n';

    std::cout << "Left: " << '\n';
    treeToString(tree->getRoot()->getLeft(), "");

    std::cout << "Horizontal: " << '\n';
    bfSearch(tree->getRoot()->getLeft());
    std::cout << '\n' << '\n';

    std::cout << "Right: " << '\n';
    treeToString(tree->getRoot()->getRight(), "");

    std::cout << "Horizontal: " << '\n';
    bfSearch(tree->getRoot()->getRight());
    std::cout << '\n' << '\n';
}

int main() {
    int action;

    std::ifstream file;
    std::string filePath;

    std::string string;

    while ((action = getAction()) != 3) {
        auto tree = new BinaryTree();

        switch (action) {
            case 1:
                std::cout << "Input your tree: ";
                std::cin >> string;
                break;
            case 2:
                std::cout << "Input the path to your file: ";
                std::cin >> filePath;

                file.open(filePath);
```

```
                if (!file.is_open()) {
                    std::cout << "Wrong file" << '\n';
                    continue;
                }

                file >> string;
                file.close();
                break;
            default:
                std::cout << "Wrong string. Try again" << '\n';
                return 0;
        }

        tree->initTree(string);

        if (!tree->getRoot()) {
            std::cout << "That's an empty tree" << "\n\n";
            continue;
        }

        execConvert(tree);
        delete tree;
    }

    return 0;
}
```