ММИНОБРНАУКИ 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: Деревья.

Студент гр. 9384 Мосин К.К.

Преподаватель Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Обработать ввод бинарного дерева. Определить глубину и путь внутреннего дерева.

Задание.

ВАРИАНТ 2д.

Для заданного бинарного дерева b типа BT с произвольным типом элементов:

- определить максимальную глубину дерева b, т. е. число ветвей в самом длинном из путей от корня дерева до листьев;

- вычислить длину внутреннего пути дерева b, т. е. сумму по всем узлам длин путей от корня до узла.

Выполнение работы.

Предполагается, что вводимые данные истины, то бишь бинарное дерево введено корректно, а также используется короткая запись. Для обозначения отсутствия узлов вводится символ ‘/’. Для создания левой ветви достаточным условием является наличие ‘(‘ символа после корня дерева. С правой ветвью гораздо сложнее. Достаточным условием является наличие символа ‘/’ после корня, либо ‘(‘ символа после четного количества ‘(‘ и ‘)’ символов. После успешного ввода дерево обрабатывается двумя функциями. Одна перебирает все узлы(лепестки), пока они не кончатся, и считает уровень, на котором находится последний узел(лепесток), тем самым получая максимальную глубину дерева. Вторая функция также перебирает узлы и только узлы, считая общую сумму глубины для каждого узла, чтобы получить путь внутреннего дерева.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

Входные данные	Выходные данные
()	NULL
(a(b(c)))	Max depth = 3 Lenght = 1
(a(b(d/(h))(e))(c(f(i)(j))(g/(k(l)))))	Max depth = 5 Lenght = 11

Выводы.

В ходе выполнения лабораторный работы было создано и обработано бинарное дерево.

ПРИЛОЖЕНИЕ
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <string>
#include <fstream>
#include <algorithm>
#include "BinaryTree.h"

using namespace std;

BinaryTree<char> *genNode(const string &, unsigned int = 1);
bool isThereRightNode(const string &, unsigned int &);
unsigned int treeDepth(BinaryTree<char> *);
unsigned int treeLenght(BinaryTree<char> *, unsigned int = 0);
void printResult(string);

int main(){
   string inputType;
   cout << "Type Console to write binary tree in console or File to read binary tree from document: ";
   cin >> inputType;
   if(inputType == "Console"){
      cout << "Type binary tree: ";
      string temp;
      cin >> temp;
      printResult(temp);
   }
   else if(inputType == "File"){
      cout << "Type binary tree document path: ";
      string temp;
      cin >> temp;
      ifstream inputFile(temp);
      if(inputFile.is_open()){
         while(getline(inputFile, temp))
            printResult(temp);
         inputFile.close();
      }
      else{
         cout << "FILE NOT OPEN" << endl;
         return -1;
      }
   }
   else{
      cout << "BAD INPUT CONFIGURATION" << endl;
      return -1;
   }
}

BinaryTree<char> *genNode(const string &string, unsigned int index){
   BinaryTree<char> *leftNode = nullptr, *rightNode = nullptr;

   if (string[index] == ')')
      return nullptr;

   if (string[index + 1] == '('
```

```
        leftNode = genNode(string, index + 2);

    unsigned int temp = index + 1;
    if (isThereRightNode(string, temp))
        rightNode = genNode(string, temp);

    return new BinaryTree<char>(string[index], leftNode, rightNode);
}

bool isThereRightNode(const string &string, unsigned int &index){
    unsigned int countBracket = 0;

    if (string[index] == '/'){
        index += 2;
        return true;
    }
    else if (string[index] == ')')
        return false;

    do{
        if (string[index] == '('
            countBracket++;
        else if (string[index] == ')')
            countBracket--;
        index++;
    }while (countBracket > 0);

    if (string[index] == '('){
        index++;
        return true;
    }
    return false;
}

unsigned int treeDepth(BinaryTree<char> *tree){
    if (tree->isNullBinaryTree())
        return 0;
    else
        return max(treeDepth(tree->getLeftNode()), treeDepth(tree->getRirghtNode())) + 1;
}

unsigned int treeLenght(BinaryTree<char> *tree, unsigned int depth){
    if(tree->isNullBinaryTree() || !tree->isNode())
        return 0;
    else
        return depth++ + treeLenght(tree->getLeftNode(), depth) + treeLenght(tree->getRirghtNode(), depth);
}

void printResult(string BT){
    BinaryTree<char> *tree = genNode(BT);
    if(!tree->isNullBinaryTree()){
        // tree->printBinaryTree();
        cout << "Max depth = " << treeDepth(tree) << " Lenght = " << treeLenght(tree) <<endl;
    }
    else
        cout << "NULL" << endl;
}
```

Название файла: binaryTree.h

```
#ifndef BINARYTREE_H
#define BINARYTREE_H

#include <iostream>

template < typename Data >
class BinaryTree{
public:
   BinaryTree(Data data, BinaryTree *leftNode, BinaryTree *rightNode){
      this->data = data;
      this->leftNode = leftNode;
      this->rightNode = rightNode;
   }

   ~BinaryTree(){
      if(!this->getLeftNode()->isNullBinaryTree())
         this->getLeftNode()->~BinaryTree();
      if(!this->getRirghtNode()->isNullBinaryTree())
         this->getRirghtNode()->~BinaryTree();
      if(!this->isNullBinaryTree())
      delete this;
   }

   void printBinaryTree(){
      if(!this->isNullBinaryTree()){
         std::cout << this->getRootBinaryTree();
         this->getLeftNode()->printBinaryTree();
         this->getRirghtNode()->printBinaryTree();
      }
   }

   bool isNullBinaryTree(){
      return this == nullptr;
   }

   Data getRootBinaryTree(){
      if(this->isNullBinaryTree()){
         error(1);
         return (Data)NULL;
      }
      else
         return this->data;
   }

   BinaryTree *getLeftNode(){
      if(this->isNullBinaryTree()){
         error(2);
         return nullptr;
      }
      else
         return this->leftNode;
   }
```

```
    BinaryTree *getRirghtNode(){
        if(this->isNullBinaryTree()){
            error(3);
            return nullptr;
        }
        else
            return this->rightNode;
    }

    bool isNode(){
        if(this->getLeftNode()->isNullBinaryTree() && this->getRirghtNode()->isNullBinaryTree())
            return false;
        return true;
    }

    void error(unsigned int err){
        switch(err){
            case 1:
                std::cout << "ERROR: IMPOSSIBLE TO GET ROOT - TREE IS NULL" << std::endl;
                break;
            case 2:
                std::cout << "ERROR: IMPOSSIBLE TO GET LEFT NODE - TREE IS NULL" << std::endl;
                break;
            case 3:
                std::cout << "ERROR: IMPOSSIBLE TO GET RIGHT NODE - TREE IS NULL" << std::endl;
                break;
            default:
                std::cout << "ERROR: SOMETHING WRONG" << std::endl;
        }
    }

private:
    Data data;
    BinaryTree *leftNode;
    BinaryTree *rightNode;
};

#endif
```