

**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: БИНАРНЫЕ ДЕРЕВЬЯ**

Студент гр. 9381

\_\_\_\_\_

Щеглов Д.А.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2021

## **Цель работы.**

Научиться работать с бинарными деревьями, обрабатывать их.

## **Задание.**

1в. Задано бинарное дерево *b* типа ВТ с типом элементов *Elem*. Для введенной пользователем величины *E* (var *E*: *Elem*):

- определить, входит ли элемент *E* в дерево *b*;
- определить число вхождений элемента *E* в дерево *b*;
- найти в дереве *b* длину пути (число ветвей) от корня до ближайшего узла с элементом *E* (если *E* не входит в *b*, за ответ принять -1

## **Выполнение работы.**

Для выполнения данной лабораторной работы необходимо было создать класс бинарного дерева. В нем содержатся поля левой и правой строки, двух чисел – двух «ссылок» на дочерние элементы в общем массиве, глубины, разделяемой строки и непосредственно того, что содержится в элементе.

Так как реализуется дерево через вектор(массив) – введем глобальный массив из 1024 элементов и последнюю позицию, в которую записывать новый элемент дерева.

Также, в классе присутствует конструктор, инициализирующий левую и правую ссылки пустыми, глубину и разделяемую строку. Помимо этого, в классе присутствуют четыре метода – разделение, создание всех дочерних веток(всего дерева), печать данного дерева и функция поиска элементов.

*Метод int divide():*

Сравниваем с конечным элементом имеющуюся разделяемую строку – в случае если это так, то это можно считать конечным листом – выходим со значением 1 – все нормально.

Проверив на длину и боковые скобки, отделив их, движемся до левого дочернего элемента. Тем самым получаем сам элемент, после чего оставшееся разделяем на левый, правый элемент, двигаясь по строке к соответствующей скобочке.

*Method int createBranches():*

Разделяем строку, если не выходит – возвращаем ноль, тем самым в главной функции выводим ошибку.

Выводим промежуточные результаты деления.

Если левый элемент – не +(не конечный). Инициализируем левую “ссылку” и создаем левый элемент, выделив под него память. Запускаем далее его создание веток на левом элементе. Если слева нет элемента – его ссылку указываем, как -1. Аналогично выполняем для правого элемента.

*Method void printTree():*

Выводим исходя из глубины отступ, сам элемент и запускаем вывод для левого и правого дерева.

*Method void findElems(std::string, int\*, int, int\*):*

Сравниваем рассматриваемый сейчас элемент с искомым, в случае если совпадают – добавляем еще +1 в переменную count, также сверяем путь bs с наименьшим путем и, если настоящий путь меньше – меняем предыдущий наименьший. Запускаем аналогичный метод у дочерних элементов, увеличив настоящий путь на 1.

В главной функции – предлагаем выбрать тип ввода и в зависимости выбора – вводим строку line из консоли или из файла.

Далее создаем голову дерева с исходной строкой и глубиной – 0. Создаем его ветки. Если не вышло – выводим ошибку.

Выводим в качестве промежуточного результата дерево.

После чего выполняем непосредственно само задание – прочитаем с ввода элемент, который нам необходимо найти. После чего выполняем метод дерева по поиску, передав в него ссылки на переменные счета элемента и минимальной длины.

В случае если таковые элементы есть(count>0) – Выводим сколько их и наименьшую длину веток, иначе – выводим, что элемент не был найден.

Разработанный программный код см. в приложении А.

## Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	(a(b)(c(d)(e))) e	(a(b)(c(d)(e))) Dividing a on (b) and (c(d)(e)) __Dividing b on + and + __Dividing c on (d) and (e) ____Dividing d on + and + ____Dividing e on + and +  a --b --c --d --e Which element would you like to find? e Element e enters the tree 1 times!Least way to this element goes through 2 branches
2.		Error!
3.	()	() Error!
4.	(a(a(z)(e(d)(c)))(a(b)(a(b)(a(b)(c)))) (a(b)(c)))) b	(a(a(z)(e(d)(c)))(a(b)(a(b)(a(b)(c)))) Dividing a on (a(z)(e(d)(c))) and (a(b)(a(b)(a(b)(c)))) __Dividing a on (z) and (e(d)(c)) ____Dividing z on + and + ____Dividing e on (d) and (c) ____Dividing d on + and + ____Dividing c on + and + __Dividing a on (b) and (a(b)(a(b)(c))) ____Dividing b on + and + ____Dividing a on (b) and (a(b)(c)) ____Dividing b on + and + ____Dividing a on (b) and (c)

		<div>_____Dividing b on + and +</div> <div>_____Dividing c on + and +</div> <div>a</div> <div>--a</div> <div>--z</div> <div>--e</div> <div>--d</div> <div>--c</div> <div>--a</div> <div>--b</div> <div>--a</div> <div>--b</div> <div>--a</div> <div>--b</div> <div>--c</div> <div>Which element would you like to find?</div> <div>b</div> <div>Element b enters the tree 3 times!Least way to this element goes through 2 branches</div>
--	--	---

### **Выводы.**

Были изучены основные методы работы с бинарными деревьями.  
 Реализована программа поиска элемента в введенном дереве.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

class BinTree{
private:
    std::string line;//Разделяемая строка
    std::string c;//Что содержится в элементе
    std::string lStr;//Левая строка
    std::string rStr;//Правая строка

    int depth=0;//Глубина в дереве

    int l;//"Ссылка" на левый элемент
    int r;//"Ссылка" на правый элемент
public:
    BinTree(std::string line, int depth);//Конструктор
    int divide();//Разделение строки
    int createBranches();//Создание всех дочерних ветвей
    void printTree();//Печать дерева
    void findElems(std::string h, int* count, int bs, int *leastbs);//Функция
    для поиска элемента
};

BinTree** arr=new BinTree*[1024];
int last=0;

BinTree::BinTree(std::string line, int depth) {
    this->line=line;
    this->depth=depth;
}

int BinTree::divide(){
    if(line.compare("+")==0)
        return 1;
    if(!(line.length()>2&&line[0]=='('&&line[line.length()-1]=='))')
        return 0;

    line=line.substr(1,line.length()-2);

    if(line[0]=='('||line[0]==')'||line[0]=='+')
        return 0;

    int i=0;

    while(line[i]!='+'&&line[i]!='(') {
        i++;
        if (i == line.length()) {
            c = line;
            lStr = "+";
            rStr = "+";
            return 1;
        }
    }
    //Содержание самого узла
    c=line.substr(0,i);

    if(line[i]=='+'){//Левая/правая часть
        lStr="+";
```

```

        i++;
        rStr=line.substr(i,line.length()-i);
    }else{
        int k=0;
        while(line[i-1]!=' '||k!=0){
            if(line[i]=='(')
                k++;
            if(line[i]==')')
                k--;
            i++;
            if(i==line.length()+1)
                return 0;
        }
        lStr=line.substr(c.length(),i-c.length());
        rStr=line.substr(i,line.length()-i);

        if(rStr.compare("")==0)
            rStr="+";
    }

    return 1;
}

int BinTree::createBranches(){

    if(divide()==0)//Если невозможно разделить
        return 0;

    for(int i=0;i<depth;i++)
        std::cout << "___";
    std::cout << "Dividing " << c <<" on " << lStr << " and " << rStr << "\n";

    if(lStr!="+") {//Создаем слева
        l=last;
        last++;
        arr[l] = new BinTree(lStr, depth+1);
        if (arr[l]->createBranches() == 0)
            return 0;
    }else
        l = -1;

    if(rStr!="+") {//Создаем справа
        r=last;
        last++;
        arr[r] = new BinTree(rStr, depth+1);
        if(arr[r]->createBranches()==0)
            return 0;
    }else
        r = -1;

    return 1;
}

void BinTree::printTree() { //Функция печати дерева

    for (int i = 0; i < depth - 1; i++)
        std::cout << "  ";
    if (depth != 0)
        std::cout << "--";
    std::cout << c << std::endl;
    if (l != -1)
        arr[l]->printTree();
    if (r != -1)
        arr[r]->printTree();
}

```

```

void BinTree::findElems(std::string elem, int* count, int bs, int *leastbs) {
    if(elem.compare(this->c)==0) {
        *count+=1;
        if(bs<*leastbs)
            *leastbs=bs;
    }
    if(this->l!=-1)
        arr[l]->findElems(elem,count,bs+1,leastbs);
    if(this->r!=-1)
        arr[r]->findElems(elem,count,bs+1,leastbs);
}

int main(){
    int ch=0;
    std::cout << "Input type:\n0.Console\n1.From file\n";
    std::cin >> ch;
    std::string line;
    if(ch==0){
        std::cin.ignore(1);
        getline(std::cin, line);
    }else if(ch==1){
        std::ifstream ff;
        ff.open("Input.txt");
        if(ff.is_open())
            getline(ff, line);
        else
            return 0;
        std::cout << "In file: " << line<< "\n";
    }

    BinTree *tree = new BinTree(line, 0);
    if(tree->createBranches()==0){
        std::cout << "Error!";
        return 0;
    };
    std::cout << std::endl;
    tree->printTree();
    int count=0;
    int leastBranches=1025;
    std::string whichElem;
    std::cout << "Which element would you like to find?\n";
    getline(std::cin,whichElem);
    tree->findElems(whichElem,&count,0,&leastBranches);
    if(count!=0){
        std::cout << "Element "<< whichElem <<" enters the tree "<<count<<"
times!";
        std::cout << "Least way to this element goes through "<<leastBranches<<"
branches";
    }else
        std::cout << "Element was not found in tree, so, amount of branches=-1";

    return 0;
}

```