

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

**ОТЧЕТ**  
**по лабораторной работе №4**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Программирование алгоритмов с бинарными деревьями**

Студент гр. 7383

\_\_\_\_\_

Александров Р.А.

Преподаватель

\_\_\_\_\_

Размочаева Н.В.

Санкт-Петербург

2018

### **Цель работы.**

Познакомиться с бинарным деревом и его реализацией на языке программирования C++.

### **Постановка задачи.**

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

- а) определить, входит ли элемент *E* в дерево *b*;
- б) определить число вхождений элемента *E* в дерево *b*;
- в) 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).

### **Реализация задачи.**

Для решения поставленной задачи в работе были использованы 3 класса: *Main*, *BT*, *Actions*.

В классе *Main* определяется функции считывания бинарного дерева:

- 1) *void fileRead()* – из файла;
- 2) *void consoleRead()* – из консоли.

Пользователю предлагается либо ввести бинарное дерево или указать текстовый файл, в котором оно находится.

В классе *BT* определяется бинарное дерево:

- *BT()* – конструктор;
- *void createBT(const string &str, int &i)* создает бинарное дерево;
- *void createRoot(const string &str, int &i)* создает корень;
- *void createChildren(const string &str, int &i, int side)* создает либо левое поддерево, либо правое поддерево;
- *bool findElem(Elem elem, BT \*bt)* ищет заданный элемент;
- *bool findOnSides(Elem elem, BT \*bt)* вспомогательная рекурсивная функция для поиска;
- *int countElems(Elem elem, BT \*bst, int &count)* считает количество вхождений элемента в дерево;

- `bool countOnSides(Elem elem, BT *bt, int &count)` вспомогательная рекурсивная функция для счета;
- `int getLeastLength(Elem elem, BT *bst)` находит кратчайший путь от корня до ближайшего узла с заданным элементом;
- `void getSidesLeastLength(Elem elem, BT *bst, int &count, bool &flag)` вспомогательная рекурсивная функция для поиска кратчайшего пути;
- `~BT()` деструктор.

В классе `Actions` определяются следующие функции:

- `void start(string str)` вызывает функции класса `BT`;
- `bool validate(string &str)` проверяет строку;
- `void showMenu()` показывает возможное меню.

### **Тестирование программы.**

Программа собрана и проверена в операционных системах Xubuntu 18.04 с использованием компилятора `g++` и Windows с использованием MinGW. В других ОС и компиляторах тестирование не проводилось. Тесты находятся в приложении А.

### **Вывод.**

В ходе лабораторной работы были получены основные навыки программирования бинарного дерева на языке `C++`. Результатом стала программа, которая находит заданный элемент в бинарном дереве, считает его количество и находит кратчайший путь от корня до него.

# **ПРИЛОЖЕНИЕ А** **РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ**

Таблица 1 – Тестирование программы

Input	Output
<p>(ab(bf123(z9)(ab))(zgm(ab)(g15)))</p> <p>Your element... z9</p> <p>Your element... ab</p>	<p>Your element [z9] was found Number of this element = 1 The least length from the root to this element = 1</p> <p>Your element [ab] was found Number of this element = 3 The least length from the root to this element = 0</p>
<p>( ab (bf123 (z9) (ab))(zgm (ab)(g15)))</p> <p>Your element... g15</p>	<p>Your btree - (ab(bf123(z9)(ab))(zgm(ab)(g15)))</p> <p>Your element [g15] was found Number of this element = 1 The least length from the root to this element = 2</p>
<p>(a((fff21)(23))</p>	<p>Wrong brackets</p>

## ПРИЛОЖЕНИЕ Б

### КОД ПРОГРАММЫ

#### **main.cpp**

```
#include <iostream>
#include <fstream>
#include <string>
#include "main.h"

using namespace std;

void Main::menu() {
    cout << "1. Enter a binary tree from the console" << endl;
    cout << "2. Enter a binary tree from the text file" << endl;
    cout << "0. Exit" << endl;
}

void Main::fileRead() {
    string fileName;
    string bt;
    cout << "What`s the file name?" << endl;
    cin >> fileName;
    cout << "-----" << endl;
    cout << "Reading from " << fileName << endl;
    cout << "-----" << endl;
    ifstream inFile;
    inFile.open(fileName);
    if (!inFile) {
        cout << "Cannot find this file" << endl;
        cout << endl;
        return;
    }
    while (!inFile.eof()) {
        getline(inFile, bt);
        actions.start(bt);
    }
    inFile.close();
}

void Main::consoleRead() {
    string btree;
    cout << "Enter btree" << endl;
    cin.ignore();
    getline(cin, btree);
    actions.start(btree);
}

int main() {
    Main main;
    while (true) {
```

```

        main.menu();
        cin >> main.choice;
        switch (main.choice) {
            case 1:
                main.consoleRead();
                break;
            case 2:
                main.fileRead();
                break;
            case 0:
                exit(1);
        }
    }
}

```

### **main.h**

```

#pragma once

#include "actions.h"

class Main {
    Actions actions;
public:

    Main() {}

    unsigned int choice;

    void fileRead();

    void consoleRead();

    void menu();
};

```

### **actions.cpp**

```

#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <vector>
#include "actions.h"

using namespace std;

void Actions::start(string btree) {
    if (!validate(btree)) {
        cout << "Wrong brackets" << endl;
    }
}

```

```

        return;
    }
    BT<string> binaryTree;
    cout << "Your btree - " << btree << endl;
    int i = 0;
    unsigned int choice;
    binaryTree.createBT(btree, i);
    while (true) {
        showMenu();
        cin >> choice;
        string element;
        int count = 0;
        switch (choice) {
            case 1:
                cout << "Your element..." << endl;
                cin >> element;
                if (binaryTree.findElem(element, &binaryTree)) {
                    cout << "-----" << endl;
                    cout << "Your element [" << element << "] was found"
<< endl;
                    cout << "Number of this element = " <<
binaryTree.countElems(element, &binaryTree, count) << endl;
                    cout << "The least length from the root to this element
= " << binaryTree.getLeastLength(element,
&binaryTree)
                    << endl;
                    cout << "-----" << endl;
                } else {
                    cout << "Your element [" << element << "] wasn't found"
<< endl;
                }
                break;
            case 0:
                return;
        }
    }
}

bool Actions::validate(string &str) {
    vector<char> brackets;
    string res;
    unsigned int countBrackets = 0;
    for (char i : str) {

```

```

        if (isspace(i)) {
            continue;
        } else if (i == '(') {
            res.push_back(i);
            countBrackets++;
            brackets.push_back(i);
        } else if (i == ')') {
            res.push_back(i);
            countBrackets++;
            try {
                brackets.pop_back();
            } catch (exception &e) {
                return false;
            }
        } else {
            res.push_back(i);
        }
    }
    str = res;
    return brackets.empty() && (countBrackets > 0);
}

```

```

void Actions::showMenu() {
    cout << "1. Enter an element" << endl;
    cout << "0. Return" << endl;
}

```

### **actions.h**

```

#pragma once

#include "btree.h"

class Actions {
public:
    void start(string str);

    bool validate(string &str);

    void showMenu();
};

```

### **btree.h**

```

#include <iostream>
#include <cstring>
#include <cctype>

```



```

#include <cstdlib>
#include <string>
#include <vector>

using namespace std;

template<typename Elem>
class BT {
private:
    enum sides {
        leftSide, rightSide
    };
    Elem element;
    BT *root;
    BT *left;
    BT *right;
public:
    BT();

    void createBT(const string &str, int &i);

    void createRoot(const string &str, int &i);

    void createChildren(const string &str, int &i, int side);

    bool findElem(Elem elem, BT *bt);

    bool findOnSides(Elem elem, BT *bt);

    int countElems(Elem elem, BT *bst, int &count);

    bool countOnSides(Elem elem, BT *bt, int &count);

    int getLeastLength(Elem elem, BT *bst);

    void getSidesLeastLength(Elem elem, BT *bst, int &count, bool &flag);

    ~BT();
};

template<typename Elem>
BT<Elem>::BT() {
    root = NULL;
    left = NULL;
    right = NULL;
}

// Init Binary Tree
template<typename Elem>
void BT<Elem>::createBT(const string &str, int &i) {
    if (str[i] == '(') {

```

```

        i++;
        createRoot(str, i);
        createChildren(str, i, leftSide);
        createChildren(str, i, rightSide);
        if (str[i] == ')') {
            i++;
        }
    } else {
        throw invalid_argument("");
    }
}

template<typename Elem>
void BT<Elem>::createRoot(const string &str, int &i) {
    string resStr;
    if (isspace(str[i + 1]) && str[i + 1] == ')') && str[i + 1] == '(') {
        resStr = str[i];
        element = resStr;
        i++;
    } else {
        while (!isspace(str[i]) && str[i] != ')') && str[i] != '(') {
            resStr.push_back(str[i]);
            i++;
        }
        element = resStr;
    }
}

template<typename Elem>
void BT<Elem>::createChildren(const string &str, int &i, int side) {
    if (str[i] != ')') {
        if (side == leftSide) {
            left = new BT;
            left->root = this;
            left->createBT(str, i);
        } else {
            right = new BT;
            right->root = this;
            right->createBT(str, i);
        }
    }
}

// function wrapper
template<typename Elem>
bool BT<Elem>::findElem(Elem elem, BT *bst) {
    if (bst == NULL) {
        return false;
    }
    if (bst->element == elem) {
        return true;
    }
}

```

```

    }
    if (!bst->left && !bst->right) {
        return false;
    }
    if (findOnSides(elem, bst->left)) {
        return true;
    }
    return findOnSides(elem, bst->right);
}

```

```

template<typename Elem>
bool BT<Elem>::findOnSides(Elem elem, BT *bst) {
    if (bst->element == elem) {
        return true;
    }
    if (bst->left) {
        if (bst->left->element == elem) {
            return true;
        }
        findOnSides(elem, bst->left);
    }
    if (bst->right) {
        if (bst->right->element == elem) {
            return true;
        }
        findOnSides(elem, bst->right);
    }
    return false;
}

```

```

// function wrapper
template<typename Elem>
int BT<Elem>::countElems(Elem elem, BT *bst, int &count) {
    if (bst == NULL) {
        return 0;
    }
    if (bst->element == elem) {
        count++;
    }
    if (bst->left) {
        if (bst->left->element == elem) {
            count++;
        }
        countOnSides(elem, bst->left, count);
    }
    if (bst->right) {
        if (bst->right->element == elem) {
            count++;
        }
        countOnSides(elem, bst->right, count);
    }
}

```

```

    }
    return count;
}

template<typename Elem>
bool BT<Elem>::countOnSides(Elem elem, BT *bst, int &count) {
    if (bst->left) {
        if (bst->left->element == elem) {
            count++;
        }
        countOnSides(elem, bst->left, count);
    }
    if (bst->right) {
        if (bst->right->element == elem) {
            count++;
        }
        countOnSides(elem, bst->right, count);
    }
    return false;
}

template<typename Elem>
int BT<Elem>::getLeastLength(Elem elem, BT *bst) {
    if (!findElem(elem, bst)) {
        return -1;
    }
    int countLeft = 0;
    int countRight = 0;
    bool flag = false; // true if we will see our element
    if (bst->left || bst->right) {
        if (bst->left->root->element == elem || bst->right->root->element
== elem) {
            return countLeft;
        }
    }
    if (bst->left) {
        if (bst->left->element == elem) {
            return ++countLeft;
        }
        getSidesLeastLength(elem, bst->left, countLeft, flag);
        if (!flag) countLeft = 0;
    }
    flag = false;
    if (bst->right) {
        if (bst->right->element == elem) {
            return ++countRight;
        }
        getSidesLeastLength(elem, bst->right, countRight, flag);
        if (!flag) countRight = 0;
    }
    if (countLeft == 0) {

```

```

        return countRight;
    }
    if (countRight == 0) {
        return countLeft;
    }
    return ((countLeft > countRight) ? countRight : countLeft);
}

```

```

template<typename Elem>
void BT<Elem>::getSidesLeastLength(Elem elem, BT *bst, int &count, bool
&flag) {
    if (bst->element != elem) {
        count++;
    }
    if (bst->left) {
        if (bst->left->element == elem) {
            flag = true;
            return;
        }
        getSidesLeastLength(elem, bst->left, count, flag);
    }
    if (bst->right) {
        if (bst->right->element == elem) {
            flag = true;
            return;
        }
        getSidesLeastLength(elem, bst->right, count, flag);
    }
}

```

```

template<typename Elem>
BT<Elem>::~~BT() {
    if (left != NULL) {
        delete left;
    }
    if (right != NULL) {
        delete right;
    }
}

```

## Makefile

```

CXX=g++
RM=rm -f
LD_FLAGS=-g -Wall

SRCS=main.cpp actions.cpp
OBJS=$(subst .cpp,.o,$(SRCS))

```

```
all: main

main: $(OBJS)
    $(CXX) $(LDFLAGS) -o main $(OBJS)

main.o: main.cpp main.h

actions.o: actions.cpp actions.h

btree.o: btree.h

clean:
    $(RM) $(OBJS)

distclean: clean
    $(RM) main
```