

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

**ОТЧЕТ**  
**по лабораторной работе №2**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Деревья**

Студент гр. 9382

\_\_\_\_\_

Герасев Г.А.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2020

## **Цель работы.**

Изучить алгоритмы работы с деревьями.

## **Задание.**

5) Написать рекурсивную функцию или процедуру, формирующую линейный список номеров всех вхождений одного бинарного коромысла в другое.

## **Основные теоретические положения.**

Коромысло – дерево без узлов, где в листьях находятся 2 числа.

## **Функции и структуры данных.**

В качестве структуры данных был использован класс, содержащий структуру, где лежат либо указатели на ветви, либо числа, виде единой структуры под названием BeanUnion. Таким образом этот класс – коромысло, для которого можно написать конструктор и различные методы.

В конструкторе рекурсивно происходит парсинг переданной строки, благодаря которому и создается структура.

В качестве основных методов можно обозначить метод проверки на равенство с другим коромыслом и метод, решающий задачу – поиск в глубину, ищущий переданное поддерево, реализованный через предыдущий метод.

## **Описание алгоритма.**

Метод применяется к некому дереву, и принимает на вход искомое дерево.

В переменных алгоритма содержатся указатели на список-результат и список, кодирующий текущий адрес в дереве.

Если метод применяется к такому же дереву, что проверяется методом проверяющим на равенство, то нынешний адрес, создаваемый в процессе работы алгоритма, записывается в результат. Иначе рассматриваются левое и правое поддерево, с соответствующим изменением нынешнего адреса.

При возврате из функции нынешний адрес стирает свой последний символ, чтобы вернуться на адрес назад, в котором и должен находиться алгоритм в данный момент.

### Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	(1,2) (1,2)	BEAMS ARE EQUAL	Невозможно дать адреса в таком случае
2.	(1,2) ((1,2), (1,2))	0 1	
3.	(1, (1,2))	Error: Invalid findMiddleComma call on string	Корректная обработка неправильного ввода
4.	(1,2) ((1,2), ((1,2),(1,2)))	THE ANSWER IS 0 10 11	Правильный ответ
5.	(1,2) ((2,2), ((1,2),(1,2)))	THE ANSWER IS 10 11	
6.	(1,3) (1,2)	THE ANSWER IS	Ответ верен, ведь список адресов – пустая строка

### Выводы.

Был изучен алгоритм обработки дерева, была создана программа, которая создает список всех вхождений одного дерева в другое.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Beam.cpp

```
#include "Beam.h"

bool isBracketValid(int bracket)
{
    return bracket != string::npos;
}

bool isPointerNull(Beam* p)
{
    return p == nullptr;
}

bool isPointersNulls(Beam* p1, Beam* p2)
{
    return (isPointerNull(p1) && isPointerNull(p2));
}

int findMiddleComma(string input)
{
    int openCounter = 0;
    int result;
    for (int i=0; i<input.length(); i++)
    {
        switch (input[i])
        {
            case '(': { openCounter++; break; }
            case ')': { openCounter--; break; }
            case ',':
            {
                if (openCounter == 1) result = i;
                break;
            }
        }
    }

    if (openCounter != 0)
    {
        throw "Error: Invalid findMiddleComma call";
        // raise(SIGILL);
        return -1; // Ugly error handle, due to the requirements
    }

    return result;
}

Beam::Beam(string inputString)
{
    if (inputString[0] != '(' && inputString[inputString.size() - 1] != ')')
```

```

{
    throw "Error: Invalid Input";
    return;
}

try
{
    int massInput = -1, leverInput = -1;
    sscanf(inputString.c_str(), "(%d,%d)", &massInput, &leverInput);
    if (!(massInput == -1 || leverInput == -1))
    {
        beamUnion.values.mass = massInput;
        beamUnion.values.lever = leverInput;
        return;
    }

    int middleComma = findMiddleComma(inputString);
    string insides = inputString.substr(1, inputString.size() - 2); // .substr(1, inputString.size() - 1)
    on StackOverflow. It's not working for some reason.

    string leftString = insides.substr(0, middleComma-1);
    string rightString = insides.substr(middleComma);

    if (!(leftString.empty() || rightString.empty()))
    {
        beamUnion.pointers.Left = new Beam(leftString);
        beamUnion.pointers.Right = new Beam(rightString);
        return;
    }

    throw "Error: Invalid Input";
    // raise(SIGILL);
}
catch (const char* msg)
{
    throw msg;
}
}

Beam::Beam(const Beam & beam) // Оператор копирования
{
    if (!isPointersNulls(beam.beamUnion.pointers.Left, beam.beamUnion.pointers.Right))
    {
        beamUnion.pointers.Left = new Beam(*beam.beamUnion.pointers.Left);
        beamUnion.pointers.Right = new Beam(*beam.beamUnion.pointers.Right);
    }

    beamUnion.values.mass = beam.beamUnion.values.mass;
    beamUnion.values.lever = beam.beamUnion.values.lever;
}

Beam::~Beam()
{
    if (!isPointerNull(beamUnion.pointers.Left))

```

```

    {
        delete (beamUnion.pointers.Left);
    }

    if (!isPointerNull(beamUnion.pointers.Right))
    {
        delete (beamUnion.pointers.Right);
    }
}

void Beam::view()
{
    if (!isPointersNulls(beamUnion.pointers.Left, beamUnion.pointers.Right))
    {
        RECURSION_DEEPNESS++;
        beamUnion.pointers.Left->view();
        beamUnion.pointers.Right->view();
        RECURSION_DEEPNESS--;
        return;
    }

    string recursionDeepnessSpaces(RECURSION_DEEPNESS*4, ' ');
    cout << recursionDeepnessSpaces << beamUnion.values.mass << ' ' << beamUnion.values.lever
    << '\n';
}

bool Beam::isEqual(Beam second)
{
    if (!isPointersNulls(beamUnion.pointers.Left, beamUnion.pointers.Right) &&
        !isPointersNulls(second.beamUnion.pointers.Left, second.beamUnion.pointers.Right))
    {
        bool isLeftEq = (*beamUnion.pointers.Left).isEqual(*second.beamUnion.pointers.Left);
        bool isRightEq = (*beamUnion.pointers.Right).isEqual(*second.beamUnion.pointers.Right);
        return (isLeftEq && isRightEq);
    }

    bool isMassEq = beamUnion.values.mass == second.beamUnion.values.mass;
    bool isLeverEq = beamUnion.values.lever == second.beamUnion.values.lever;
    return (isMassEq && isLeverEq);
}

bool Beam::isContains(Beam second)
{
    if (this->isEqual(second))
    {
        return true;
    }

    if (!isPointersNulls(beamUnion.pointers.Left, beamUnion.pointers.Right))
    {
        bool isInLeft, isInRight;
        isInLeft = beamUnion.pointers.Left->isContains(second);
        isInRight = beamUnion.pointers.Right->isContains(second);
        return (isInLeft || isInRight);
    }
}

```

```

    }
    return false;
}

void Beam::isContainsListHandler(Beam second, string* result_ptr, string* currentlyAt_ptr)
{
    if (this->isEqual(second))
    {
        result_ptr->append(*currentlyAt_ptr);
        result_ptr->append(" ");
        *currentlyAt_ptr = currentlyAt_ptr->substr(0, currentlyAt_ptr->size() -1);
        return;
    }

    if (!isPointersNulls(beamUnion.pointers.Left, beamUnion.pointers.Right))
    {
        currentlyAt_ptr->append("0");
        beamUnion.pointers.Left->isContainsListHandler(second, result_ptr, currentlyAt_ptr);

        currentlyAt_ptr->append("1");
        beamUnion.pointers.Right->isContainsListHandler(second, result_ptr, currentlyAt_ptr);
    }

    *currentlyAt_ptr = currentlyAt_ptr->substr(0, currentlyAt_ptr->size() -1);
    return;
}

string* Beam::isContainsList(Beam second)
{
    auto result_ptr = new string;
    auto currentlyAt_ptr = new string;
    this->isContainsListHandler(second, result_ptr, currentlyAt_ptr);
    return result_ptr;
}

void Beam::isContainsListHandlerWithOutput(Beam second, string* result_ptr, string*
currentlyAt_ptr, int deepness)
{
    string buffer(deepness, ' ');

    if (this->isEqual(second))
    {
        cout << buffer << "Found the tree\n";
        result_ptr->append(*currentlyAt_ptr);
        result_ptr->append(" ");
        *currentlyAt_ptr = currentlyAt_ptr->substr(0, currentlyAt_ptr->size() -1);
        cout << buffer << "Go up, out this node\n";
        return;
    }

    if (!isPointersNulls(beamUnion.pointers.Left, beamUnion.pointers.Right))
    {
        currentlyAt_ptr->append("0");
        cout << buffer << "Go left\n";

```

```

    beamUnion.pointers.Left->isContainsListHandlerWithOutput(second, result_ptr,
currentlyAt_ptr, ++deepness);

    currentlyAt_ptr->append("1");
    cout << buffer << "Go right\n";
    beamUnion.pointers.Right->isContainsListHandlerWithOutput(second, result_ptr,
currentlyAt_ptr, ++deepness);
}

cout << buffer << "Go up, out this node\n";
*currentlyAt_ptr = currentlyAt_ptr->substr(0, currentlyAt_ptr->size() -1);;
return;
}

string* Beam::isContainsListWithOutput(Beam second)
{
    auto result_ptr = new string;
    auto currentlyAt_ptr = new string;
    cout << "Call the handler function\n";
    this->isContainsListHandlerWithOutput(second, result_ptr, currentlyAt_ptr, 0);
    return result_ptr;
}

void inputHandler(string whatToSearch, string whereToSearch)
{
    Beam beam1(whatToSearch);
    Beam beam2(whereToSearch);

    if (beam1.isEqual(beam2))
    {
        cout << "BEAMS ARE EQUAL\n";
    }
    else
    {
        string result = *(beam2.isContainsListWithOutput(beam1));
        cout << "THE ANSWER IS " << result << "\n";
    }
}

void introductionMessageView()
{
    cout << "File input example: ./main -f test.txt\n\n";
    cout << "Separator -- <, >, do not use space\n";
    cout << "Examples of trees: (any positive int numbers can be in brackets)\n";
    cout << "(1,1)\n";
    cout << "((1,1),(1,1))\n";
    cout << "((1,1),(((2,1),(1,1)),(1,1)))\n";
    cout << "((((1,1),(1,1)),((1,1),(1,1))),(((1,1),(1,1)),((1,1),(1,1))))\n\n";
    cout << "q to exit\n";
}

void stdInputCase()
{
    string whatToSearch, whereToSearch;

```



```

cin >> whatToSearch;
cin >> whereToSearch;

while (whatToSearch.compare("q") != 0)
{
    try
    {
        Beam beam1(whatToSearch);
        Beam beam2(whereToSearch);
        inputHandler(whatToSearch, whereToSearch);
    }
    catch (const char* msg)
    {
        cerr << msg << '\n';
    }

    cout << "Please, input binary tree to search and where to search.\n";
    cin >> whatToSearch;
    cin >> whereToSearch;
}
}

void fileInputCase(string path)
{
    ifstream inFile;
    inFile.open(path);

    string whatToSearch, whereToSearch;

    while (inFile >> whatToSearch && inFile >> whereToSearch)
    {
        try
        {
            Beam beam1(whatToSearch);
            Beam beam2(whereToSearch);
            inputHandler(whatToSearch, whereToSearch);
        }
        catch (const char* msg)
        {
            cerr << msg << '\n';
        }
    }
    inFile.close();
}

int main(int argc, char *argv[])
{
    if (argc >= 2)
    {
        string flag(argv[1]);
        string path(argv[2]);
        if (flag.compare("-f") == 0)
            fileInputCase(path); // No obvious way to overload the function
        return 0;
    }
}

```

```

    }
    cout << "Please, input binary tree to search and where to search.\n";
    introductionMessageView();
    stdInputCase();

    return 0;
}

```

Название файла: Beam.h

```
#pragma once
```

```
#include <iostream>
```

```
#include <string.h>
```

```
#include <csignal>
```

```
#include <fstream>
```

```
using namespace std;
```

```
int RECURSION_DEEPNESS = 0;
```

```
class Beam
```

```
{
```

```
private:
```

```
    struct Values { int mass = -1, lever = -1; };
```

```
    struct Pointers
```

```
{
```

```
    Beam* Left = nullptr;
```

```
    Beam* Right = nullptr;
```

```
};
```

```
struct BeamUnion
```

```
{
```

```
    Values values;
```

```
    Pointers pointers;
```

```
    // No way to make structure like this : (Values || Pointers)
```

```
    // Union is too low-level, and there is no way to tell the difference
```

```
    // between Values and Pointers
```

```
};
```

```
BeamUnion beamUnion;
```

```
void isContainsListHandler(Beam second, string* result_ptr, string* currentlyAt);
```

```
void isContainsListHandlerWithOutput(Beam second, string* result_ptr, string* currentlyAt_ptr, int  
deepness);
```

```
public:
```

```
Beam(string inputString = "(0, 0)");
```

```
Beam(const Beam & beam);
```

```
~Beam();
```

```
void view();
```

```
bool isEqual(Beam second);
```

```
bool isContains(Beam second);
```

```
string* isContainsList(Beam second);
```

```
string* isContainsListWithOutput(Beam second);
```

```
};
```

```
bool isPointerNull(Beam* p);
```

```
bool isPointersNulls(Beam* p1, Beam* p2);
```

```
bool isBracketValid(int bracket);
```