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:Деревья

Студент гр. 9304 Прокофьев М.Д.

Преподаватель Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Узнать о деревьях и о их использовании в практике

Задание.

Рассматриваются бинарные деревья с элементами типа Elem (в качестве Elem использовать char). Заданы перечисления узлов некоторого дерева b в порядке КЛП и ЛКП. Требуется: - восстановить дерево b и вывести его изображение; - перечислить узлы дерева b в порядке ЛПК

Выполнение работы.

Для выполнения работы был создан класс *BinTree*, являющийся деревом, функция *drawTree()* для рисования дерева, функция *printTree()* – для написания дерева в ЛПК форме и функция *repair()*.

В функции repair происходит восстановление дерева по ЛКП и КЛП. Функция принимает две строки: заданные ЛКП и КЛП записи дерева, итератор i для обхода по КЛП и итератор *old_i* для обхода по ЛКП. Функция работает по следующему алгоритму: Происходит последовательный “побуквенный” цикл по строке КЛП: при просмотре буквы в КЛП, происходит поиск этой же буквы в ЛКП, причем поиск начинается сначала вправо, потом, если нужно - влево от номера предыдущего результата поиска. Соответственно, если поиск шел слева направо, то при нахождении нужного элемента, тот записывается в правую ветку, иначе в левую. Здесь функция *No_old()* служит для того, чтобы прерывать цикл, если в поиске обнаруживается та буква, которая была уже просмотрена ранее. При удачном поиске буквы происходит не только запись в дерево, но и рекурсивный вызов этой же функции *repair*, таким образом происходит полное восстановление дерева.

В случае, если ЛКП и КЛП записи заданы неправильно, алгоритм не проходит полностью и, поэтому, выводится “Tree is incorrect”. В случае, если дерево задано правильно, то, во-первых, выводится его изображение с помощью функции *drawTree()*, а во-вторых, выводится запись в ЛПК форме.

Тестирование.

Был написан python-скрипт для проведения тестов(testing.py). Посредством testing.py, (“входной” строке в программе) последовательно передаются строки из файлов SuccessTests.txt и ErrorTests.txt. Таким образом, при “запуске” Makefile, программа обрабатывает 2 строки из вышеуказанных файлов(SuccessTests.txt и ErrorTests.txt), после чего выдает ответы в консоли.

Результаты тестирования изложены в приложении В.

Выводы.

Написана программа с использованием дерева. Кроме того, в программе использовались умные указатели. Использование дерева при написании программы являлось необходимым, поскольку в задании целью является восстановление дерева.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: main.cpp

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

class BinTree{
public:
    char value;
    std::shared_ptr<BinTree> left {nullptr};
    std::shared_ptr<BinTree> right {nullptr};
};



void drawTree(std::shared_ptr<BinTree> curNode, int level, int direction)
{
    if(curNode)
    {
        drawTree(curNode->right,level + 1, 1);
        for(int i = 0;i< level;i++) std::cout<<"    ";
        if(direction == 1) std::cout << "/ ";
        else if(direction == 2) std::cout << "\\ ";
        std::cout << curNode->value << std::endl;
        drawTree(curNode->left,level + 1, 2);
    }
}
```

```
void printTree(std::shared_ptr<BinTree> curNode) {
    if (!curNode) {
        return;
    }
    printTree(curNode->left);
    printTree(curNode->right);
    std::cout << curNode->value << ' ';
}

bool No_old(char a, char b, char* KLP_local)
{
    for(int i=0; KLP_local[i]&&(KLP_local[i]!=b); i +=2)
        if(KLP_local[i]==a) return false;
    return true;
}

void repair(std::shared_ptr<BinTree> current_node, char *LKP, char* KLP, int
&i_main, int old_i)
{
    int i;
    i_main +=2;
    char exec;
    for(int j=0; j<2; j++)
    {
        exec = KLP[i_main];
        if(current_node->right==nullptr)
        for(i=old_i+2; (LKP[i])&&No_old(LKP[i], exec, KLP); i +=2)
        {
            if(exec == LKP[i])
            {
```

```
            current_node->right = std::make_shared<BinTree>();
            current_node->right->value = exec;
            repair(current_node->right, LKP, KLP, i_main, i);
            break;
        }
      }
      if(current_node->left==nullptr)
      for(i=old_i-2; ((i>-2))&&No_old(LKP[i], exec, KLP); i -=2)
      {
        if(exec == LKP[i])
        {
            current_node->left = std::make_shared<BinTree>();
            current_node->left->value = exec;
            repair(current_node->left, LKP, KLP, i_main, i);
            break;
        }
      }
   }
}



int main(int argc, char* argv[])
{
   std::string input1, input2;
   std::ifstream in("input.txt");
   getline(in, input1);
   getline(in, input2);
   char KLP[50];
   char LKP[50];
```

```
    strcpy(KLP, input1.c_str());
    strcpy(LKP, input2.c_str());

    if(strlen(KLP)==strlen(LKP))
    {
    int i_glav = 0;
    int i = 0;
    std::shared_ptr<BinTree> My_Tree=std::make_shared<BinTree>();
    My_Tree->value=KLP[0];
    for(i=0; LKP[i]&&(LKP[i]!=My_Tree->value);  i +=2);
    repair(My_Tree, LKP, KLP,  i_glav, i);
    if((i_glav-1)==strlen(KLP)) {
                        std::cout << "Picture of tree:\n";
                        drawTree(My_Tree, 0, 0);
                        std::cout << "Tree in LPK:\n";
                        printTree(My_Tree);
                        std::cout << std::endl;
                        }
    else std::cout << "Tree is incorrect";
    } else std::cout << "Tree is incorrect";
    return 0;
}
```

ПРИЛОЖЕНИЕ В
ТЕСТИРОВАНИЕ

Результаты тестирования:

Первое тестирование:

Входные данные:

a b c d e f g

c b d a f e g

Выходные данные:

Picture of tree:

```
      / g
   / e
      \ f
a
      / d
   \ b
      \ c
```

Tree in LPK:

c d b f g e a

Комментарий:

Обычное бинарное дерево.

Второе тестирование:

Входные данные:

a b c d e f g h i j k
b a d c g f h e j i k

Выходные данные:

Picture of tree:

```
            / k
          / i
            \ j
        / e
            / h
          \ f
            \ g
    / c
      \ d
a
    \ b
```

Tree in LPK:
b d g h f j k i e c a

Комментарий:
Длинное бинарное дерево с отсутствующими ветками

Третье тестирование:

Входные данные:

a b c d e

a b c d e

Выходные данные:

Picture of tree:

```
        / e
      / d
    / c
  / b
a
```

Tree in LPK:

e d c b a

Комментарий:

Бинарное дерево только “с правыми ветками”

Четвертое тестирование:

Входные данные:

a b d f c e g

f d b a c e g

Выходные данные:

Picture of tree:

```
        / g
      / e
    / c
a
    \ b
      \ d
        \ f
```

Tree in LPK:

f d b g e c a

Комментарий:

Бинарное дерево ветви которой идут только либо в правую сторону, либо в левую.

Пятое тестирование:

Входные данные:

a

a

Выходные данные:

Picture of tree:

a

Tree in LPK:

a

Комментарий:

Бинарное дерево состоящее из 1 элемента

Шестое тестирование:

Входные данные:

a b c d e f g

b c d a r g h

Выходные данные:

Tree is incorrect

Комментарий:

Бинарное дерево с несовпадающими элементами, соответственно оно неверное.

Седьмое тестирование:

Входные данные:

a b c d e f g
c b d a f e

Выходные данные:

Tree is incorrect

Комментарий:

Бинарное дерево с недостающими элементами, соответственно оно неверное.

Восьмое тестирование:

Входные данные:

a (b c d e f) g
c b d a f e g

Выходные данные:

Tree is incorrect

Комментарий:

Бинарное дерево с неверно расставленными скобками, соответственно оно неверное.

Девятое тестирование:

Входные данные:

a b c d e f g p o l k o p r s
c b d a f e g

Выходные данные:

Tree is incorrect

Комментарий:

Бинарное дерево с избытком элементов в КЛП записи, соответственно оно неверное.

Десятое тестирование:

Входные данные:

|||
||||

Выходные данные:

Tree is incorrect

Комментарий:

Некорректная запись дерева.