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: Алгоритмы обработки бинарных деревьев

Студент гр. 9381 _____ Аухадиев А.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Познакомиться со структурой данных бинарного дерева, реализовать класс бинарных деревьев и методы для его обработки на языке программирования C++.

Задание.

Вариант 10д.

Рассматриваются бинарные деревья с элементами типа *Elem* (в качестве *Elem* использовать *char*). Заданы перечисления узлов некоторого дерева *b* в порядке ЛКП и ЛПК. Требуется:

- восстановить дерево *b* и вывести его изображение;
- перечислить узлы дерева *b* в порядке КЛП.

Основные теоретические положения.

Дерево – конечное множество *T*, состоящее из одного или более узлов, таких, что

а) имеется один специально обозначенный узел, называемый *корнем* данного дерева;

б) остальные узлы (исключая корень) содержатся в $m \geq 0$ попарно не пересекающихся множествах T_1, T_2, ..., T_m, каждое из которых, в свою очередь, является деревом. Деревья T_1, T_2, ..., T_m называются *поддеревьями* данного дерева.

Бинарное дерево – конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом.

Описание алгоритма.

1) Создание бинарного дерева.

Метод createBinTree класса BinTree принимает две строки: запись бинарного дерева в ЛПК- и ЛКП-формате. Последний символ в строке ЛПК является корнем, после сохранения которого символ удаляется из строки. После этого совершается поиск этого корня в ЛКП-формате записи дерева. Если символ не может быть найден, значит, запись дерева была сделана неверно.

Корень делит строку ЛКП-формата на две части, левая часть которого

будет находиться слева от корня, а правая - справа. Затем данный метод рекурсивно вызывается для правого сына дерева с обрезанной строкой-записью ЛПК-формата и правой частью от найденного символа строки-записи ЛКП-формата. Затем метод вызывается для левого сына с обрезанной строкой ЛПК и с левой частью строки ЛКП от корня.

2) Вывод дерева в КЛП-формате.

В методе printTreeKLP происходит вывод корня, затем метод рекурсивно вызывается сначала для левого, затем за правого сына.

Описание структур и классов.

Класс BinTree.

Класс описывает бинарное дерево с использованием шаблона template<typename Elem> для информации, хранимой в узле дерева.

Приватные поля:

1) Elem info_ - информация, хранимая в корне дерева;

2) BinTree* left_ и BinTree* right_ - указатели на левый и правый сыновей дерева соответственно;

Публичные методы:

1) Конструктор;

2) Конструктор, принимающий значение в корне;

3) Геттеры для получения полей;

4) Сеттеры для установки значений полей;

5) BinTree* createBinTree(std::string& LPK, const std::string LKP, int n) - рекурсивный метод для восстановления бинарного дерева из ЛПК-формата LPK и ЛКП-формата LKP. LPK передаётся по ссылке, так как после каждого рекурсивного выполнения метода у LPK должен удаляться последний символ - полученный корень. Число n используется для вывода табуляции для визуализации работы рекурсии.

6)void printTreeKLP() - происходит вывод корня, затем метод рекурсивно вызывается сначала для левого, затем за правого сына.

7)void printTree(int n, int flagRight = 0) - рисование дерева в консоли в горизонтальном положении ("на левом боку"). Метод рекурсивно выводит значения узлов дерева справа налево с выводом табуляции и символов "\" и "/" для понятной визуализации.

8)Деструктор - рекурсивно очищает выделенную память в порядке ЛПК.

Разработанный программный код см. в приложении А.

Тестирование.

Выходные данные:

№	Входные данные	Выходные данные
1.	dfebgihca dbfeagchi	Корень: a, Правое поддерево: gchi, Левое поддерево:dbfe Анализ правого поддерева gchi Корень: c, Правое поддерево: hi, Левое поддерево:g Анализ правого поддерева hi Корень: h, Правое поддерево: i, Левое поддерево: Анализ правого поддерева i Корень: i, Правое поддерево: , Левое поддерево: Анализ правого поддерева завершён Анализ правого поддерева завершён Анализ левого поддерева g Корень: g, Правое поддерево: , Левое поддерево: Анализ левого поддерева завершён Анализ правого поддерева завершён Анализ левого поддерева dbfe Корень: b, Правое поддерево: fe, Левое поддерево:d

		Анализ правого поддерева fe Корень: e, Правое поддерево: , Левое поддерево:f Анализ левого поддерева f Корень: f, Правое поддерево: , Левое поддерево: Анализ левого поддерева завершён Анализ правого поддерева завершён Анализ левого поддерева d Корень: d, Правое поддерево: , Левое поддерево: Анализ левого поддерева завершён Анализ левого поддерева завершён КЛП-формат: abdefcghi
2.	abc cba	Корень: c, Правое поддерево: ba, Левое поддерево: Анализ правого поддерева ba Корень: b, Правое поддерево: a, Левое поддерево: Анализ правого поддерева a Корень: a, Правое поддерево: , Левое поддерево: Анализ правого поддерева завершён Анализ правого поддерева завершён КЛП-формат: cba
3.	dfebgihca dbfepgchi	Введённые данные некорректны
4.	dfebgihca dbfeag	Введённые данные некорректны
5.	a	Корень: a, Правое поддерево: , Левое поддерево:

	a	КЛП-формат: a
6.		Введены пустые строки

Выводы.

Была изучена структура данных бинарного, реализована рекурсивная обработка бинарных деревьев на языке программирования C++.

Разработан класс бинарного дерева *BinTree*, при помощи которого можно создать бинарное дерево по его записи в ЛПК и ЛКП - форматах. При реализации методов класса *BinTree* использовалась рекурсия.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

template<typename Elem>
class BinTree{
private:
    Elem info_;
    BinTree *left_;
    BinTree *right_;
public:
    BinTree(): left_(nullptr), right_(nullptr){};
    BinTree(Elem info):info_(info), left_(nullptr), right_(nullptr){}
//getters:
    Elem getInfo(){ return info_;}
    BinTree* getLeft(){return left_;}
    BinTree* getRight(){return right_;}
//setters
    void setInfo(Elem info){ info_ = info;}
    void setLeft(BinTree* left){left_ = left;}
    void setRight(BinTree* right){right_ = right;}

    BinTree<Elem>* createBinTree(std::string& LPK, const std::string LKP,
int n){ //рекурсвный метод восстановления дерева
        if(LPK.empty()) //условие выхода из рекурсии
            return nullptr;
        char c = LPK[LPK.length()-1];           //сохранение последнего
символа
        BinTree *binTree = new BinTree<Elem>(c);//ЛПК-формата - корня
        LPK.erase(LPK.length()-1,1);     //удаление последнего символа из
ЛПК-формата
        int index = LKP.find(c);                  //Поиск символа в ЛКП-
формате
        if(index == -1){                          //Если символ не найден
            std::cout << "Введённые данные некорректны\n";
            exit(0);
        }
        std::string right = LKP.substr(index+1);
        std::string left  = LKP.substr(0, index);
        printTab(n);
        std::cout << "Корень: " << binTree->getInfo() <<", "
                  << "Правое поддерево: " <<  right << ", "
                  << "Левое поддерево:" << left << '\n';
        if(!right.empty()){
//создание правого сына из
            printTab(n);
            std::cout << "Анализ правого поддерева " << right << '\n';
            binTree->setRight(createBinTree(LPK, right, n+1)); //строки из
ЛКП-формата,
            printTab(n);
            std::cout << "Анализ правого поддерева завершён\n";
        }
//стоящей справа от index

        if(!left.empty()) {
//содание левого сына из
            printTab(n);
            std::cout << "Анализ левого поддерева " << left << '\n';
```

```
                binTree->setLeft(createBinTree(LPK, left, n+1)); //строки из
ЛКП-формата,
                printTab(n);
                std::cout << "Анализ левого поддерева завершён\n";
            }                                                                  //
стоящей слева от index

            return binTree;
        }

        void printTreeKLP(){    //Вывод в КЛП-формате
            std::cout << this->getInfo();
            if(left_)
                left_->printTreeKLP();
            if(right_)
                right_->printTreeKLP();
        }

        void printTree(int n, int flagRight = 0){   //Вывод дерева, "лежащего
на левом боку"
            if(right_)
                right_->printTree(n+4, 1);
            for(auto i = 0; i<n; i++)
                std::cout << " ";
            std::cout << info_ << "\n";
            if(flagRight || left_) {
                for (auto i = 0; i < n + 1; i++) {
                    if (i == n - 3 && flagRight)
                        std::cout << '/';
                    if (i == n && left_)
                        std::cout << "\\";
                    std::cout << ' ';
                }
                std::cout << '\n';
            }
            if(left_)
                left_->printTree(n+4);
        }

        void printTab(int n){
            for(auto i = 0; i<n; i++)
                std::cout <<'\t';
        }

        ~BinTree(){
            if(left_)
                delete left_;
            if(right_)
                delete right_;
        }
    };

    int main() {
        std::string LPK;
        std::string LKP;
        std::cout << "Выберите формат ввода: 0 - консоль, 1 - файл\n";
        int choise;
        std::cin >> choise;
        switch(choise){
            case 0:
                std::cout << "Введите строку в ЛПК-формате\n";
                std::cin >> LPK;
                std::cout << "Введите строку в ЛКП-формате\n";
                std::cin >> LKP;
```

```
            break;
        case 1: {
            std::string fileName;
            std::cout << "Введите название файла\n";
            std::cin >> fileName;
            std::ifstream file(fileName);
            if (!file.is_open()) {
                std::cerr << "Файл не найден\n";
                return 0;
            }
            file >> LPK;
            file >> LKP;
            std::cout << "ЛПК: " << LPK << "\nЛКП: " << LKP << '\n';
            file.close();
            break;
        }
        default:
            std::cerr << "Неверный формат ввода\n";
            return 0;
    }
    if(LKP.empty() || LPK.empty()) {
        std::cerr << "Введены пустые строки\n";
        return 0;
    }
    if(LKP.length() != LPK.length()){
        std::cerr << "Введённые данные некорректны\n";
        return 0;
    }
    BinTree<char>* binTree;
    binTree = binTree->createBinTree(LPK, LKP, 0);
    std::cout << "\nКЛП-формат: ";
    binTree->printTreeKLP();
    std::cout << "\nРисунок дерева:\n";
    binTree->printTree(0);
    delete binTree;
    return 0;
}
```