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

ОТЧЕТ

по практической работе №3

по дисциплине «Вычислительная математика»

Тема: Деревья

Студентка гр. 9303 _____ Отмахова М.А.

Преподаватель _____ Сучков А.И.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с определением бинарного дерева и научиться программировать структуру данных «бинарное дерево».

Основные теоретические положения.

Дадим формальное определение.

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 0 попарно не пересекающихся множествах T 1 , T 2 , ..., T m , каждое из которых, в свою чередь, является деревом. Деревья T 1 , T 2 , ..., T m называются поддеревьями данного дерева.

При программировании и разработке вычислительных алгоритмов удобно использовать именно такое рекурсивное определение, поскольку рекурсивность является естественной характеристикой этой структуры данных.

Каждый узел дерева является корнем некоторого поддерева. В том случае, когда множество поддеревьев такого корня пусто, этот узел называется концевым узлом, или листом. Уровень узла определяется рекурсивно следующим образом: 1) корень имеет уровень 1; 2) другие узлы имеют уровень, на единицу больший их уровня в содержащем их поддереве этого корня.

Говорят, что каждый корень является отцом корней своих поддеревьев и что последние являются сыновьями своего отца и братьями между собой.

Говорят также, что узел n – предок узла m (а узел m – потомок узла n), если n– либо отец m, либо отец некоторого предка m.

Постановка задачи.

Вариант 4в.

Для заданного бинарного дерева b типа BT с произвольным типом элементов определить, есть ли в дереве b хотя бы два одинаковых элемента.

Выполнение работы.

Для реализации самого бинарного дерева создаем класс `BinTree` и структуру `Node` для описания каждого узла дерева. В структуре `Node` есть три поля – `right` и `left`, а также поле `data` для хранения значения.

Класс бинарного дерева (`BinTree`) содержит такие поля: поле `Node* vec` – это массив, в котором содержатся все узлы дерева, а `size` – это максимально позволенный размер этого массива.

Методы класса `BinTree`:

`void resize()` – метод для увеличения размера `vec`.

`void addNode(T data, int ind)` – метод для добавления элемента в `vec`.

`bool doubleChecker(int counter)` – метод, который проверяет вектор на наличие одинаковых элементов.

Функции:

`void readBinTree(string& infix, int start, int end, int ind, BinTree<char>* bt, int& counter)` – функция для считывания дерева в инфиксной записи.

В функции main()

Исходный код программы представлен в приложении А.

Тестирование программы представлено в приложении Б.

Выводы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

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

using namespace std;

template <typename T>
class BinTree{
private:
    struct Node{
        T data;
        int left = 0;
        int right = 0;
    };
    int size;
public:
    Node* vec;
    BinTree(){
        vec = new Node [100];
        size = 100;
    }

    void resize(){
        int newSize = size + 100;
        Node* tmp = new Node [newSize];
        for (int i = 0; i<size; i++)
            tmp[i] = vec[i];
        delete [] vec;
        size = newSize;
        vec = tmp;
    }

    void addNode(T data, int ind){
        if (ind*2+2 == size)
            this->resize();
        vec[ind].data = data;
```

```
            vec[ind].left = ind*2 + 1;
            vec[ind].right = ind*2 + 2;
        }

        bool doubleChecker(int counter){
            int id = 0;
            int num = 0;
            while (num < counter){
                if(isalnum(vec[id].data)) {
                    for(int i = 0; i < id; i++) {
                        if (vec[i].data == vec[id].data)
                            return true;
                    }
                    num++;
                }
                id++;
            }
            return false;
        }

        ~BinTree(){
            delete [] vec;
        }
    };

    void readBinTree(string& infix, int start, int end,
int ind, BinTree<char>* bt, int& counter){
        int depth = 0;
        if (start == end){
            bt->addNode(infix[start], ind);
            counter++;
            return;
        }
        for (int i = start; i<=end; i++){
            if (infix[i] == '(')
                depth++;
            if (infix[i] == ')')
                depth--;
```

```
            if (depth == 1 && (infix[i] == '+' ||
infix[i] == '-' || infix[i] == '*')){
                bt->addNode(infix[i], ind);
                readBinTree(infix, start+1, i-1, ind*2+1,
bt, counter);
                readBinTree(infix, i+1, end-1, ind*2+2,
bt, counter);
            }
        }
    }

    int main(){
        string infix;
        int counter = 0;
        ifstream
fin("C:\\Users\\Masha\\CLionProjects\\untitled3\\input.tx
t");
        while(!fin.eof()){
            BinTree <char>* bt = new BinTree <char>;
            fin >> infix;
            cout << "infix: " << infix << '\n';
            readBinTree(infix, 0, infix.length()-1, 0,
bt, counter);
            cout << '\n';
            if (bt->doubleChecker(counter)) {
                cout << "True";
            } else {
                cout << "False";
            }
            delete bt;
        }
        fin.close();
        return 0;
    }
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Результаты тестирования программы представлено в таблице 1.

Таблица 1 - Тестирование программы

Входные данные	Результат работы программы
(A+(A*C))	True
(A*(B+(C*D)))	False
(A+(A-B))	True
(B+(C-(A-D)))	False
(B+(C-(A-B)))	True