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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Бинарные деревья

Студент гр. 9304 _____ Алексеенко Б.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

Создания базового функционала для работы с бинарным деревом, а также написания функции-перобразования для бинарного дерева формулы

Задание.

Вариант 14:

- преобразовать дерево-формулу t, заменяя в нем все поддеревья, соответствующие формулам $(f_1 * (f_2 + f_3))$ и $((f_1 + f_2) * f_3)$, на поддеревья, соответствующие формулам $((f_1 * f_2) + (f_1 * f_3))$ и $((f_1 * f_3) + (f_2 * f_3))$.

Основные теоретические положения.

Дадим формальное определение *дерева*, следуя [10].

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.

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

Выполнение работы.

Бинарное дерево в памяти было представлено в виде вектора, для это было реализовано два шаблонных класса: binTree и node, где первый хранит само дерево и методы, для работы с ним, а второй – представление одного узла.

Методы для работы с деревом:

- binTree() – конструктор класса, который выделяет память под массив node.
- resize() – функция, которая увеличивает размер массива, если это нужно.

- restructring() - функция, которая изменяет дерево так, как это требуется в задании.
- treeInput() – функция, которая инициализирует дерево из файла.
- addElem() - функция, которая добавляет новый элемент.
- isFornula() - функция, которая проверяет подходит ли выражение под заданную формулу.
- detour() - функция, которая делает обход по дереву.
- setVal() - функция для установки нового значения в узле.
- print() - функция для вывода формулы.

Выводы.

Во время выполнения лабораторной работы были реализованы классы, которые представляют структуру бинарного дерева. Были реализованы методы для его обработки, вывода и преобразования, было проведено тестирование.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

main.cpp

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

using namespace std;

template <typename T>
class binTree {
private:
    struct node {
        T data;
        bool head = false;
        int leftIndex = 0;
        int rightIndex = 0;
    };

public:
    int size = 0;
    node* tree;
    binTree();
    void resize();
    void restructring(int index);
    void treeInput(string& str, int start, int end, int index);
    void addElem(T data, int index);
    bool isFormula(int index_);
    void detour(int index, int& start);
    void setVal(int index, T value);
    void print(int index, int k, int& start);


};

template<typename T>
binTree<T>::binTree() {
    this->tree = new node[16];
    this->size = 16;
}


template<typename T>
void binTree<T>::addElem(T data, int index) {

    if (index * 2 + 2 >= this->size) {
        this->resize();
    }
    this->tree[index].data = data;
    if (!isalnum(data)) {
        this->tree[index].leftIndex = index * 2 + 1;
```

```
        this->tree[index].rightIndex = index * 2 + 2;
    }
}


template<typename T>
void binTree<T>::resize() {
    int newSize = this->size * 2;
    node* temp = new node[newSize];
    for (int i = 0; i < this->size; i++) {
        temp[i] = this->tree[i];
    }

    delete[] this->tree;
    this->size = newSize;
    this->tree = temp;
}

template<typename T>
void binTree<T>::treeInput(string& str, int start, int end, int
index) {
    int depth = 0;

    if (start == end) {
        this->addElem(str[start], index);
        return;
    }

    for (int i = start; i <= end; i++) {

        if (str[i] == '(')
            depth++;
        if (str[i] == ')')
            depth--;

        if (depth == 1 && (str[i] == '+' || str[i] == '*')) {
            this->addElem(str[i], index);
            this->treeInput(str, start + 1, i - 1, index * 2 + 1);
            this->treeInput(str, i + 1, end - 1, index * 2 + 2);
        }
    }
}

template <typename T>
void binTree<T>::setVal(int index, T value) {
    if (index || this->tree[index].head) {
        this->tree[index].data = value;
    }
    else {
        this->addElem(value, index);
    }
}
```

```
template <typename T>
void binTree<T>::restructring(int index) {
    setVal(index, '+');
    //cout << "curr: " << index << endl;
    if (this->tree[index * 2 + 1].data == '+') {
        setVal( (index * 2 + 2) * 2 + 2 , this->tree[index * 2 +
2].data );
        setVal( index * 2 + 1 , '*');
        this->tree[index * 2 + 2].leftIndex = (index * 2 + 2) * 2
+ 2;
        setVal( (index * 2 + 2) * 2 + 1 , this->tree[(index * 2 +
1) * 2 + 2].data );
        setVal( (index * 2 + 1) * 2 + 2 , this->tree[(index * 2 +
2) * 2 + 2].data );
        this->tree[index * 2 + 2].rightIndex = (index * 2 + 2) * 2
+ 1;
        setVal(index * 2 + 2, '*');
        swap(this->tree[2 * index + 2].leftIndex, this->tree[2 *
index + 2].rightIndex);
    }

    else {

        setVal( (index * 2 + 1) * 2 + 1 , this->tree[index * 2 +
1].data);
        setVal( index * 2 + 1 , '*');
        this->tree[index * 2 + 1].leftIndex = (index * 2 + 1) * 2
+ 1;
        setVal( (index * 2 + 1) * 2 + 2 , this->tree[(index * 2 +
2) * 2 + 1].data);
        setVal( (index * 2 + 2) * 2 + 1 , this->tree[(index * 2 +
1) * 2 + 1].data);
        this->tree[index * 2 + 1].rightIndex = (index * 2 + 1) * 2
+ 2;
        setVal( index * 2 + 2 , '*');

    }
}


template<typename T>
void binTree<T>::print(int index, int k, int& start) {
    if (index || start) {
        start = 0;
        if (k == 1) {
            cout << "(";
        }
        print(this->tree[index].leftIndex, 1, start);

        cout << this->tree[index].data;

        print(this->tree[index].rightIndex, 2, start);
        if (k == 2) {
```

```
            cout << ")";
        }
    }
    else {
        return;
    }
}

template <typename T>
bool binTree<T>::isFormula(int index_) {
    if (this->tree[this->tree[index_].leftIndex].data == '+' &&
this->tree[index_].rightIndex && isalnum(this->tree[this-
>tree[index_].rightIndex].data)) {

        if (this->tree[this->tree[index_].leftIndex].leftIndex &&
this->tree[this->tree[index_].leftIndex].rightIndex) {
            return true;
        }
        else {
            return false;
        }
    }

    else if (this->tree[this->tree[index_].rightIndex].data == '+'
&& this->tree[index_].leftIndex && isalnum(this->tree[this-
>tree[index_].leftIndex].data)) {
        if (this->tree[this->tree[index_].rightIndex].leftIndex &&
this->tree[this->tree[index_].rightIndex].rightIndex) {
            return true;
        }
        else {
            return false;
        }
    }

    else {
        return false;
    }
}

template<typename T>
void binTree<T>::detour(int index, int& start) {
    if (index != 0 || start) {
        start = 0;
        this->detour(this->tree[index].leftIndex, start);
        if ((this->tree[index].data == '*') && (this-
>isFormula(index))) {
            this->restructring(index);
        }
        this->detour(this->tree[index].rightIndex, start);
    }
    else {
        return;
```

```
    }
}

int main() {

    string infix;
    ifstream fin("input.txt");
    while (!fin.eof()) {
        binTree<char>* bt = new binTree<char>;
        fin >> infix;
        cout << "input: " << infix << '\n';
        bt->treeInput(infix, 0, infix.length()-1, 0);
        bt->tree[0].head = true;
        int start = 1;
        bt->detour(0, start);
        start = 1;
        bt->print(0, 0, start);
        cout << endl << endl;
        /*
        for (int i = 0; i < bt->size; i++)
        {
            cout << i << ": " << bt->tree[i].data << " | " << bt-
>tree[i].leftIndex << " | " << bt->tree[i].rightIndex << endl;
        }
        */
        delete bt;
    }
    fin.close();
    return 0;
}
```

ПРИЛОЖЕНИЕ А

ТЕСТИРОВАНИЕ

Таблица 1 – Примеры тестовых случаев.

№	Входные данные	Выходные данные	Комментарии
1.	(a*(b+c))	((a*b)+(a*c))	Программа работает корректно.
2.	((b+c)*a)	((b*a)+(c*a))	Программа работает корректно.
3.	((a+b)+(c+d))	(((a*b)+(a*c))+((a*c)+(b*c)))	Программа работает корректно.
4.	((a*(b+c))+((a+b)*c))	(((a*b)+(a*c))+((a*c)+(b*c)))	Программа работает корректно.
5.	((a+b)*(c+d))	((a+b)*(c+d))	Программа работает корректно.
6.	(((d+(a*(b+c)))+(d*(g*(f+h))))+((v*x)*j))	(((d+((a*b)+(a*c)))+(d*((g*f)+(g*h))))+((v*x)*j))	Программа работает корректно.