

**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: Бинарные деревья**

Студент гр. 9304

Шуняев А.В.

Преподаватель

Филатов А.Ю.

Санкт-Петербург

2020

### Цель работы.

Ознакомиться с понятием бинарного дерева, реализовать его, решить поставленную задачу на его основе

### Задание.

Вариант 13м.

Формулу вида:

$\langle \text{формула} \rangle ::= \langle \text{терминал} \rangle \mid (\langle \text{знак} \rangle \langle \text{формула} \rangle \langle \text{формула} \rangle)$

$\langle \text{знак} \rangle ::= + \mid - \mid *$

$\langle \text{терминал} \rangle ::= \mid ( ) ::= + \mid - \mid * ::= 0 \mid 1 \mid \dots \mid 9 \mid a \mid b \mid \dots \mid z$

можно представить в виде бинарного дерева.

- формула из одного терминала представляется деревом из одной вершины с этим терминалом;
- формула вида  $(s \ f1 \ f2)$  представляется деревом, в котором корень – это знак  $s$ , а левое и правое поддеревья – соответствующие представления формул  $f1$  и  $f2$ .

Требуется:

- построить дерево-формулу  $t$  из строки, задающей формулу в постфиксной форме (перечисление узлов  $t$  в порядке ЛПК);
- упростить дерево-формулу  $t$ , выполнив в нем все операции вычитания, в которых уменьшаемое и вычитаемое – цифры. Результат вычитания – цифра или формула вида  $(0 - \text{цифра})$ .

### Описание алгоритма работы программы.

На вход программы подаются строки неограниченной длины. Данные строки записываются в список строк. Далее в цикле, пока список не пуст, создается объект класса `BTree`, конструктор которого принимает строку как параметр. В нем инициализируются поля класса, и вызывается метод `TreeInit`. Данный метод рекурсивно преобразует строку в бинарное дерево на основе массива по принципу, если  $i$ -тый элемент это корень, то  $2*i+1$  его левое ветвление, а  $2*i+2$  – правое. Если элемент строки является символом «+» «-»

«\*» «/», то метод записывает его в массив и вызывается рекурсивно сначала для правого ветвления, а затем для левого. Если элемент массива является цифрой или буквой, то они записываются в массив. После создания объекта дерево выводится в два файла output.txt, output\_test.txt. После этого вызывается метод Simplify, который заменят все вычитания, в которых фигурируют цифры, на результат вычитания. После этого дерево снова выводится в два файла.

### **Формат входных и выходных данных**

Входные данные представлены в виде строк. В них входят константы и переменный, а также символы «+» «-» «\*» «/». Формула записывается в постфиксной форме. Пример «ab+cd-\*». Строки считываются из файла input.txt.

### **Описание основных структур данных и функций (кратко).**

- Класс TreeElement – элемент дерева, мы создаем массив элементов этого класса.
- Класс InputData – считывает входные данные из файла.
- Класс BTree – класс самого дерева.
- Метод TreeInit – создает дерево из строки.
- Метод Print и PrintTest – выводят дерево в два файла.
- Метод Simplify – производит упрощение вычитаний.

### **Тестирование**

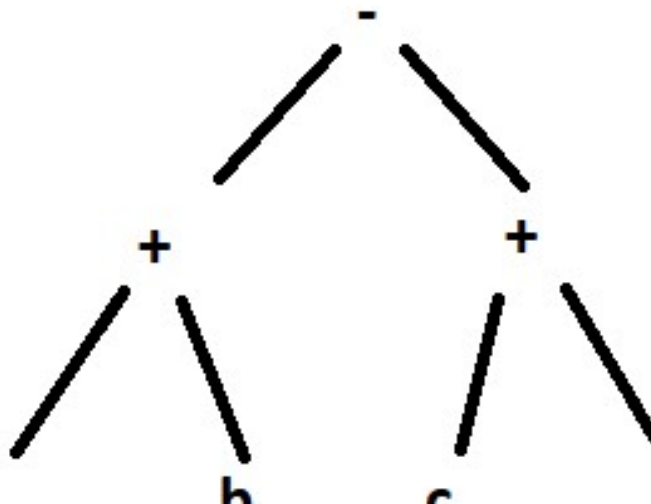
Тестирование происходит с помощью сравнения файлов output\_test.txt, true\_output.txt

Таблица 1 – тестирование программы

Входные данные	Выходные данные	Результат теста
----------------	-----------------	-----------------

ab+cd+-	((a+b)-(c+d))	Program answer: ((a+b)-(c+d)) True answer: ((a+b)-(c+d)) Test #1 - True!
ab+54-*	((a+b)*(5-4))	Program answer: ((a+b)*(5-4)) True answer: ((a+b)*(5-4)) Test #2 - True!
ab+54-*	((a+b)* 1 )	Simplified Program answer: ((a+b)* 1 ) True answer: ((a+b)* 1 ) Test #3 - True!
12+34+-56/78*+*	(( ( 1 + 2 ) - ( 3 + 4 ) ) * (( 5 / 6 ) + ( 7 * 8 ) ) )	Program answer: (( ( 1 + 2 ) - ( 3 + 4 ) ) * ( ( 5 / 6 ) + ( 7 * 8 ) ) ) True answer: (( ( 1 + 2 ) - ( 3 + 4 ) ) * ( ( 5 / 6 ) + ( 7 * 8 ) ) ) Test #4 - True!

Представление бинарного дерева в программе на примере ab+cd+-:



Примеры вывода в файлы на примере ab+cd+-:

Output\_test.txt:

-----< Tree #1 >-----

(( a + b ) - ( c + d ) )

-----< Simplified Tree #1 >-----

(( a + b ) - ( c + d ) )

Output.txt:

-----< Tree #1 >-----

-

+

a

b

+

c

d

-----< Simplified Tree #1 >-----

-

+

a

b

+

c

d

## **Выводы.**

Было изучено понятие бинарного дерева. Был реализован бинарное дерево на основе массива на языке программирования C++.

Во время выполнения работы была написана программа, создающая бинарное дерево и упрощающая вычитания. Проведено тестирование программы. В ходе выполнения работы был сделан вывод, что бинарные деревья достаточно трудно преобразовать из строки в массив, однако запись такого формата легко обрабатывать, и мы имеем доступ к любому элементу дерева, если знаем только его индекс. В то же время, у такой реализации есть существенный минус, она занимает много места и при этом не использует все, из-за чего появляется большое количество пустых ячеек.

## ПРИЛОЖЕНИЕ А

### КОД ПРОГРАММЫ.

Файл mail.cpp:

```
#include <memory>
#include <string>
#include <math.h>
#include "TreeElement.h"
#include "InputData.h"

template <typename T1>
class BTree
{
public:
    int tree_size = 0;
    std::unique_ptr<TreeElement<T1>[]> tree;

    BTree(std::string& str);
    ~BTree();

    void Print(int tab, int iter, std::ofstream& file);
    void PrintTest(int iter, std::ofstream& test_file);
    void Simplify();
    void TreeInit(std::string& str, int tree_iter, int& str_iter);

};

template<typename T1>
BTree<T1>::BTree(std::string& str)
{
    int tree_iter = 0;
    int str_iter = str.length() - 1;
    int temp = 0;

    this->tree_size = 16;
    this->tree = std::make_unique<TreeElement<T1>[]>(this->tree_size);

    this->TreeInit(str, tree_iter, str_iter);
}

template<typename T1>
BTree<T1>::~~BTree()
{
}
```

```

template<typename T1>
void BTree<T1>::Print(int tab, int iter, std::ofstream& file)
{
    int temp = tab;
    std::string str = "";
    while (temp != 0) {
        str += " ";
        temp--;
    }
    if (tree[iter].is_set) {
        if (std::holds_alternative<T1>(this->tree[iter].data_)) {
            if (std::get<T1>(this->tree[iter].data_) < 0) {
                file << str << "0 - " << abs(std::get<T1>(this->tree[iter].data_)) << '\n';
            }
            else {
                file << str << std::get<T1>(this->tree[iter].data_) << '\n';
            }
        }
        else {
            file << str << std::get<char>(this->tree[iter].data_) << '\n';

            if (std::get<char>(this->tree[iter].data_) == '+' || std::get<char>(this->tree[iter].data_) == '-' ||
                std::get<char>(this->tree[iter].data_) == '*' || std::get<char>(this->tree[iter].data_)
== '/')
            {
                this->Print(tab + 1, (2 * iter) + 1, file);
                this->Print(tab + 1, (2 * iter) + 2, file);
            }
        }
    }
}

template<typename T1>
void BTree<T1>::PrintTest(int iter, std::ofstream& test_file)
{
    if (tree[iter].is_set) {
        if (std::holds_alternative<T1>(this->tree[iter].data_)) {
            if (std::get<T1>(this->tree[iter].data_) < 0) {
                test_file << "0 - " << abs(std::get<T1>(this->tree[iter].data_)) << '\n';
            }
            else {
                test_file << std::get<T1>(this->tree[iter].data_) << '\n';
            }
        }
    }
}

```



```

else {
    //file << str << std::get<char>(this->tree[iter].data_) << '\n';

    if (std::get<char>(this->tree[iter].data_) == '+' || std::get<char>(this->tree[iter].data_) == '-' ||
        std::get<char>(this->tree[iter].data_) == '*' || std::get<char>(this->tree[iter].data_)
== '/')
    {
        test_file << "(" ";
        this->PrintTest((2 * iter) + 1, test_file);
        test_file << std::get<char>(this->tree[iter].data_) << "'";
        this->PrintTest((2 * iter) + 2, test_file);
        test_file << ") ";
    }
    else {
        test_file << std::get<char>(this->tree[iter].data_) << "'";
    }
}

}

template<typename T1>
void BTree<T1>::Simplify()
{
    for (int i = this->tree_size - 1; i >= 0; --i) {
        if (tree[i].is_set) {
            if (std::holds_alternative<char>(this->tree[i].data_) && std::get<char>(this->tree[i].data_) ==
'-' )
            {
                if (std::holds_alternative<int>(this->tree[(2 * i) + 1].data_)
                    && std::holds_alternative<int>(this->tree[(2 * i) + 2].data_))
                {
                    tree[i].data_ = std::get<int>(this->tree[(2 * i) + 1].data_) -
std::get<int>(this->tree[(2 * i) + 2].data_);

                    tree[(2 * i) + 1].is_set = false;
                    tree[(2 * i) + 2].is_set = false;
                }
            }
        }
    }
}

template<typename T1>
void BTree<T1>::TreeInit(std::string& str, int tree_iter, int& str_iter)
{
    if (tree_iter >= this->tree_size - 2) {

        int new_size = tree_size * 2;

```

```

std::unique_ptr<TreeElement<T1>[]> temp = std::make_unique<TreeElement<T1>[]>(new_size);

for (int i = 0; i < tree_size; i++) {
    temp[i] = tree[i];
}

this->tree = std::move(temp);
this->tree_size = new_size;
}

bool right_side = true;
for (str_iter; str_iter >= 0; --str_iter)
{
    if (str[str_iter] == '+' || str[str_iter] == '-' || str[str_iter] == '*' || str[str_iter] == '/')
    {
        if (str_iter == str.length() - 1) {
            tree[tree_iter].data_ = str[str_iter];
            tree[tree_iter].is_set = true;
        }
        else {
            if (right_side) {
                tree[(2 * tree_iter) + 2].data_ = str[str_iter];
                tree[(2 * tree_iter) + 2].is_set = true;
                this->TreeInit(str, 2 * (tree_iter)+2, --str_iter);
                right_side = false;
            }
            else {
                tree[(2 * tree_iter) + 1].data_ = str[str_iter];
                tree[(2 * tree_iter) + 1].is_set = true;
                this->TreeInit(str, 2 * (tree_iter)+1, --str_iter);
                right_side = true;
            }
            if (right_side) {
                return;
            }
        }
    }
    else {
        if ((str[str_iter] >= 'a' && str[str_iter] <= 'z') || (str[str_iter] >= '0' && str[str_iter] <= '9'))
        {
            if (right_side) {
                if ((str[str_iter] >= '0' && str[str_iter] <= '9')) {
                    char s = str[str_iter];
                    T1 temp = std::atoi(&s);
                    tree[(2 * tree_iter) + 2].data_ = temp;
                    tree[(2 * tree_iter) + 2].is_set = true;
                    right_side = false;
                }
            }
        }
    }
}

```

```

else {
    tree[(2 * tree_iter) + 2].data_ = str[str_iter];
    tree[(2 * tree_iter) + 2].is_set = true;
    right_side = false;
}
}
else {
    if ((str[str_iter] >= '0' && str[str_iter] <= '9')) {
        char s = str[str_iter];
        T1 temp = std::atoi(&s);
        tree[(2 * tree_iter) + 1].data_ = temp;
        tree[(2 * tree_iter) + 1].is_set = true;
        right_side = true;
    }
    else {
        tree[(2 * tree_iter) + 1].data_ = str[str_iter];
        tree[(2 * tree_iter) + 1].is_set = true;
        right_side = true;
    }
}

if (right_side) {
    return;
}

}

}

}

```

```

        file << "\n\n -----< Simplified Tree #" << iter << " >----- \n\n";
        test_file << "\n\n -----< Simplified Tree #" << iter << " >----- \n\n";

        b_tree.Print(0,0, file);
        b_tree.PrintTest(0, test_file);

        data.list.pop_front();
        iter++;
    }

    test_file.close();
    file.close();
    return 0;
}

```

Файл TreeElement.h:

```

#pragma once
#include <variant>

template <typename T1>
class TreeElement
{
public:
    bool is_set = false;
    std::variant<T1, char> data_;

};

```

Файл InputData.h:

```

#pragma once
#include <string>
#include <list>
#include <fstream>
#include <iostream>

class InputData
{
public:
    std::list<std::string> list;
    std::ifstream file_;

    InputData();

};

```

Файл InputData.cpp:

```

#include "InputData.h"

```

```

InputData::InputData()
{
    file_.open("./Tests/input.txt");
    std::string str;

    if (file_.is_open()) {
        while (std::getline(file_, str)) {
            std::cout << str << "\n";
            this->list.push_back(str);
        }
    }
    else {
        std::cout << "Can't open input file!" << std::endl;
    }

    file_.close();
}

```