

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

ОТЧЕТ
по лабораторной работе №2
по дисциплине «Алгоритмы и структуры данных»
Тема: Деревья

Студент гр. 9382

Бочаров Г.С.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Освоить некоторые методы работы с деревьями. Научиться представлять логические выражения с помощью дерева. Вычислить результат логического выражения с помощью дерева и рекурсии.

Задание.

18) логическое, вычисление, добавить 4-ую операцию (которая может принимать 2 аргумента), префиксная форма.

Основные теоретические положения.

Пусть выражение (логическое, арифметическое, алгебраическое*) представлено иерархическим списком. В выражение входят константы и переменные, которые являются атомами списка. Операции представляются в префиксной форме ((<операция> <аргументы>)), либо в постфиксной форме (<аргументы> <операция>). Аргументов может быть 1, 2 и более. Например (в префиксной форме): (+ a (* b (- c))) или (OR a (AND b (NOT c))).

В задании даётся один из следующих вариантов требуемого действия с выражением: проверка синтаксической корректности, упрощение (преобразование), вычисление.

Пример упрощения: (+ 0 (* 1 (+ a b))) преобразуется в (+ a b).

В задаче вычисления на входе дополнительно задаётся список значений переменных

((x1 c1) (x2 c2) ... (xk ck)),

где x_i – переменная, а c_i – её значение (константа).

В индивидуальном задании указывается: тип выражения (возможно дополнительно - состав операций), вариант действия и форма записи. Всего 9 заданий.

* - здесь примем такую терминологию: в арифметическое выражение входят операции +, -, *, /, а в алгебраическое — +, -, * и дополнительно некоторые функции.

Функции и структуры данных.

Структура Node {std::string value, std::vector<Node>, childrens, Type type}
хранит строковое значение элемента дерева, список его детей и тип.

Структура Var хранит имя и значение переменной.

ReadVariables() -Функция преобразует строку в которой заданы имена и значения переменных в массив токенов, для удобства работы с выражением. Также проверит корректность данного выражения

getToken() - Функция преобразует строку в массив токенов, для удобства работы с выражением

getOperator() Функция обрабатывает встреченный оператор и его аргументы

getSentence() Функция обрабатывает скобки и простые выражения

getVarValue() -Функция возвращает значение переменной, принимая на вход имя переменной и массив заданных переменных. Если значение переменной не было найдено выводит сообщение об ошибке

doAction() - Функция выполняет требуемую операцию, также проверяет корректность имен операторов

Calculate() - Функция рекурсивно вычисляет значение логического выражения, проходя по дереву в глубину и вычисля значения узлов.

Описание алгоритма.

На вход программе подается 2 выражение. Первое — задает имена и значения переменных, второе — задает само логическое выражение.

Для удобства работы с рекурсией оба выражения были разбиты на токены.

Создается массив структур Var для хранения заданных переменных.

С помощью двух взаимно рекурентных функций создается дерево, узлами которого являются операторы, а листьями — переменные и константы.

Используя обход дерева в глубину, вычисляются значения (результат выполнения той или иной логической операции) в узлах дерева, и в корне.

Таким образом вычисляется значение всего выражения.

Также с помощью обхода в глубину данное дерево выводится на экран.

Графическое представление дерева для выражения $(- (+ a (^ 0 c)))$ показано на рисунке 1 в Приложении Б.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	$((a \quad 1) \quad (b \quad 0))$ $(+ (- a) (- b))$	$+$ $-$ a $-$ b $--> res = 1$	Вывод дерева и результата

2.	$((a \ 1) \ (b \ 0) \ (c \ 1))$ $(- (+ a (^ 0 c)))$	- + a ^ 0 c --> res = 0	Вывод дерева и результата.
3.	$((x1 \ 0) \ (x2 \ 0) \ (x3 \ 1))$ $(* (- x1) (- (+ x2 x1)))$	--> res = 1	
4.	$(+ 1 (- 0))$	--> res = 1	Обработка выражения без переменных
5.	$((a \ 1) \ (b \ 0) \ (c \ 1))$ $(- (+ a (^ 0 G)))$	--> res = Значение переменной <G> не указано	Значение переменной не указано
6.	$((a \ 7) \ (23a \ 0) \ (7 \ b))$ $(- (+ a (^ 0 c)))$	Недопустимое значение переменной <a>	Значение переменной не является нулем или единицей
7.	1	--> res = 1	Выражение из 1 элемента
8.	$((23a \ 1) \ (1 \ 1))$ $(+ 1 \ 1)$	Недопустимое имя переменной <23a>	Некорректное имя переменной

Выводы.

Был изучен алгоритм обработки вычисления логического выражения в префиксной форме. Освоены приемы работы с деревьями.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
```

```
#include <vector>
```

```
#include <sstream>
```

```
#include <fstream>
```

```
#include <string>
```

```
enum Type {
```

```
    OPERATOR, VAR
```

```
};
```

```
// Структура данных для хранения оператора, переменной или константы
```

```
struct Node {
```

```
    std::string value;
```

```
    std::vector<Node> childrens;
```

```
    Type type;
```

```
};
```

```
// Структура данных для хранения переменной (имя + значение)
```

```
struct Var {
```

```
    std::string name; //имя переменной
```

```
    int val; //значение переменной
```

```
};
```

```
//Функция преобразует строку в которой заданы имена и значения переменных  
в массив токенов, для удобства работы с выражением
```

```
//также проверит корректность данного выражения
```

```
template<typename IterT>
```

```
void readVariables(std::vector<Var> *res, IterT &first, const IterT &last) {
```

```

if (first >= last) {
    return;
}

if (*first == "(") {
    if (*(first + 3) != ")")
        throw std::runtime_error("Неправильный формат ввода5");

    readVariables(res, ++first, last);

} else if (*first == ")") {

    if (*(first - 3) != "(")
        throw std::runtime_error("Неправильный формат ввода");
    readVariables(res, ++first, last);
} else {
    if ((*first + 1) != "1") && (*first + 1) != "0")
        throw std::runtime_error("Недопустимое значение переменной <" + *first +
">");
    if (!isalpha(*first)) {
        throw std::runtime_error("Недопустимое имя переменной <" + *first +
">");
    }
    Var var = {*first, stoi(*first + 1)};
    res->push_back(var);
    if (first + 2 >= last) {
        throw std::runtime_error("Неправильный формат ввода");
    } else {
        first += 2;
    }
    if (*first != ")")

```

```

        throw std::runtime_error("Неправильный формат ввода");
    readVariables(res, first, last);
    return;
}
}

```

//Ф-ция выводит попарно имена и значения для каждой переменной

```

void printVariables(std::vector<Var> *res) {
    for (auto &i : *res) {
        std::cout << "name : " << i.name << std::endl;
        std::cout << "value : " << i.val << std::endl;
    }
}

```

//Функция преобразует строку в массив токенов, для удобства работы с выражением

```

template<typename StreamT>
std::vector<std::string> getToken(StreamT &stream) {
    std::vector<std::string> result;
    std::string temp;
    char symbol = 0;

    std::string line;
    std::getline(stream, line);
    line.push_back('\n');
    std::istringstream str(line);
    str >> std::noskipws;
    while (symbol != '\n') {
        str >> symbol;
        if ((symbol == ' ') || ((symbol == '\n')) {
            if (!temp.empty())

```



```

        result.push_back(temp);
        temp.clear();
    }
    if ((symbol == '(') || (symbol == ')')) {

        if (!temp.empty())
            result.push_back(temp);
        temp.clear();
        temp.push_back(symbol);
        result.push_back(temp);
        temp.clear();
    }
    if ((symbol != '(') && (symbol != ')') && (symbol != ' ') && (symbol != '\n'))
        temp.push_back(symbol);
    }
    return result;
}

```

//Ф-ция проверяет, является ли оператор унарным

```

bool isUnoOperator(const std::string &s) {
    if (s == "-")
        return true;
    return false;
}

```

//Ф-ция проверяет, является ли оператор бинарным

```

bool isBinOperator(const std::string &s) {
    if ((s == "+") || (s == "*") || (s == "^"))
        return true;
    return false;
}

```

```

template<typename IterT>
void getSentence(IterT &first, const IterT &last, Node *&parent);

//Ф-ция заносит оператор в дерево и считывает его аргументы
template<typename IterT>
void getOperator(IterT &first, const IterT &last, Node *&parent) {

    if (first >= last) {
        return;
    }

    std::vector<Node> tmp;
    Node atom = {*first, tmp, OPERATOR};

    Node *next;

    if (parent == nullptr) {
        parent = new Node();
        parent->value = *first;
        parent->childrens = tmp;
        parent->type = OPERATOR;
        next = parent;
    } else {
        parent->childrens.push_back(atom);
        next = &parent->childrens.back();
    }

    if (isBinOperator(*first)) {

        getSentence(++first, last, next);
    }
}

```

```
    getSentence(++first, last, next);
```

```
  } else {
```

```
    getSentence(++first, last, next);
```

```
  }
```

```
}
```

//Ф-ция рекурсивно обрабатывает скобки в выражении. Если выражение состоит из 1 части (например "1, 0, a, b ..."), заносит его в дерево

```
template<typename IterT>
```

```
void getSentence(IterT &first, const IterT &last, Node *&parent) {
```

```
    if (first >= last) {
```

```
        return;
```

```
    }
```

```
    if (*first == "(") {
```

```
        getOperator(++first, last, parent);
```

```
    } else if (*first == ")") {
```

```
        getSentence(++first, last, parent);
```

```
    } else {
```

```
        std::vector<Node> tmp;
```

```
        Node atom = {*first, tmp, VAR};
```

```
        if (parent == nullptr) {
```

```
            parent = new Node();
```

```
            parent->value = *first;
```

```
            parent->childrens = tmp;
```

```
            parent->type = VAR;
```

```
        } else {
```

```
            parent->childrens.push_back(atom);
```

```
        }
```

```
        return;
```

```
    }
```

```
}
```

//Ф-ция выводит построенное дерево на экран

```
void widthPrint(Node *parent, int &k) {  
    std::string sp(k, '\t');  
    std::cout << sp << parent->value << std::endl;  
    k++; // Глубина элемента  
    for (auto &i:parent->childrens) {  
        widthPrint(&i, k);  
        k--;  
    }  
}
```

//Ф-ция возвращает значение переменной, принимая на вход имя переменной и массив заданных переменных. Если значение переменной не было найдено выводит сообщение об ошибке

```
bool getVarValue(Node *a, const std::vector<Var> &Vars) {  
    for (auto &i:Vars)  
        if (i.name == a->value)  
            return i.val;  
    throw std::runtime_error("Значение переменной <" + a->value + "> не  
указано");  
}
```

//Ф-ция выполняет требуемую операцию, также проверяет корректность имен операторов

```
bool doAction(const std::string &opName, bool a, bool b = false) {  
  
    if (opName == "+")  
        return a || b;  
    else if (opName == "*")
```

```

        return a && b;
    else if (opName == "^")
        return a ^ b;
    else if (opName == "-")
        return !a;
    else
        throw std::runtime_error("Неопознанный оператор <" + opName + ">");
}

```

//Ф-ция рекурсивно вычисляет значение логического выражения, проходя по дереву в глубину и вычисляя значения узлов.

```

bool Calculate(Node *parent, std::vector<Var> &Variables) {
    if (parent->type == VAR) {
        if (parent->value == "1")
            return true;
        else if (parent->value == "0")
            return false;
        else return getVarValue(parent, Variables);
    } else if (isUnoOperator(parent->value)) {
        if (parent->childrens.size() != 1)
            throw std::runtime_error("Неверное количество аргументов у <" + parent->value + ">");
        else
            return doAction(parent->value, Calculate(&parent->childrens.at(0), Variables));
    } else if (parent->childrens.size() != 2)
        throw std::runtime_error("Неверное количество аргументов у <" + parent->value + ">");
    else
        return doAction(parent->value, Calculate(&parent->childrens.at(0), Variables), Calculate(&parent->childrens.at(1), Variables));
}

```

```
}
```

```
int main() {
```

```
    try {
```

```
        while (1) {
```

```
            Node *head = nullptr;
```

```
            std::vector<std::string> TokensVar; // Массив токенов для выражения,  
задающего имена и значения переменных
```

```
            std::vector<std::string> TokensSentence; // Массив токенов для логического  
выражения
```

```
            std::vector<Var> Variables;
```

```
            int readFormat;
```

```
            std::cout << "0 - считать из файла, 1 - считать с консоли" << std::endl;
```

```
            std::cin >> readFormat;
```

```
            std::cin.ignore();
```

```
            switch (readFormat) {
```

```
                case 0: {
```

```
                    std::cout << "Введите имя файла : ";
```

```
                    std::ifstream in;
```

```
                    std::string fileName;
```

```
                    std::cin >> fileName;
```

```
                    in.open(fileName);
```

```
                    if (in) {
```

```
                        TokensVar = getToken(in);
```

```
                        TokensSentence = getToken(in);
```

```

    } else
        throw std::runtime_error("Файл не найден!");
    in.close();
    break;
}
case 1: {
    std::cout<<"Введите имена переменных и их значения. Например :
((a 0) (x 1))"<<std::endl;
    TokensVar = getToken(std::cin);

    std::cout<<"Введите выражение в префиксной форме. Например : (+
a (* 0 (- x)))"<<std::endl;
    TokensSentence = getToken(std::cin);
    break;
}
default:
    throw std::runtime_error("Неветный формат ввода");
}

if (!TokensVar.empty()) {
    if ((*TokensVar.begin() != "(") || (*(TokensVar.end() - 1) != "))")
        throw std::runtime_error("Неправильный формат ввода");
    auto beg = TokensVar.begin() + 1, last = TokensVar.end() - 1;
    readVariables(&Variables, beg, last);
}

auto beg = TokensSentence.begin(), last = TokensSentence.end();
getSentence(beg, last, head);

int action;

```

```

std::cout << " 1 - вывести значения переменных, \n"
        " 2 - вывести дерево,\n"
        " 3 - вывести результат вычислений,\n"
        " 0 - выход" << std::endl;

while ((std::cin >> action) && (action != 0)) {
    switch (action) {
        case 1: {
            printVariables(&Variables);
            break;
        }
        case 2: {
            int k = 0;
            widthPrint(head, k);
            break;
        }
        case 3: {
            std::cout << "--> res = " << Calculate(head, Variables) << std::endl;
            break;
        }
        case 0: {
            break;
        }
        default:
            std::cout << "Выбрано неверное действие" << std::endl;
            break;
    }
}

```

```

int temp;

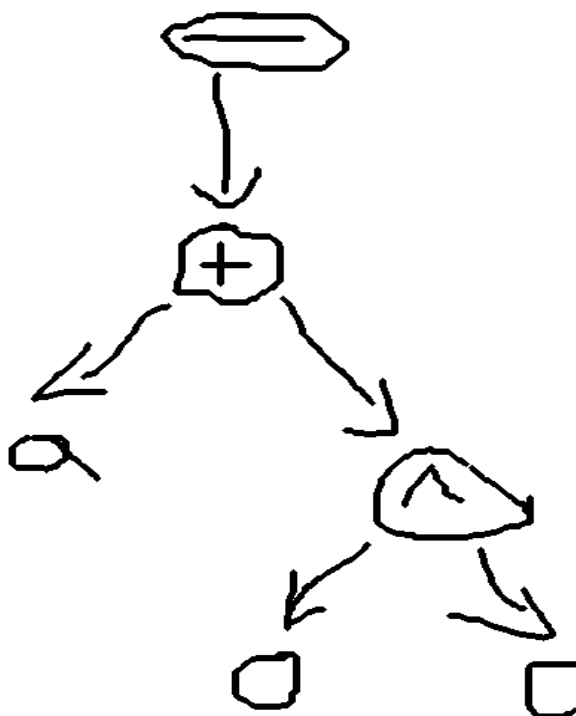
std::cout << "Для повторного ввода нажмите 1, для выхода - любую

```



```
другую клавишу" << std::endl;
    std::cin >> temp;
    std::cin.ignore();
    if(temp != 1)
        break;
}
} catch (std::exception &e) {
    std::cerr << e.what() << std::endl;
    return 0;
}
}
```

ПРИЛОЖЕНИЕ Б



(Рисунок 1)